Europäisches Patentamt

(19) European Patent Office

Office européen des brevets

(11) EP 0 680 652 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention
of the grant of the patent:
**08.09.1999 Bulletin 1999/36**

(21) Application number: **94907854.7**

(22) Date of filing: **18.01.1994**

(51) Int Cl.$^6$: **G10L 3/00, G10L 5/02,
G10L 7/02, G10L 9/14**

(86) International application number:
**PCT/US94/00770**

(87) International publication number:
**WO 94/17517 (04.08.1994 Gazette 1994/18)**

(54) **WAVEFORM BLENDING TECHNIQUE FOR TEXT-TO-SPEECH SYSTEM**

WELLENFORM-MISCHUNGSVERFAHREN FÜR SYSTEM ZUR TEXT-ZU-SPRACHE
UMSETZUNG

TECHNIQUE DE MELANGE DE FORMES D'ONDES POUR SYSTEME DE CONVERSION
TEXTE-PAROLE

(84) Designated Contracting States:
**DE ES FR GB**

(30) Priority: **21.01.1993 US 7621**

(43) Date of publication of application:
**08.11.1995 Bulletin 1995/45**

(73) Proprietor: **APPLE COMPUTER, INC.
Cupertino, CA 95014 (US)**

(72) Inventor: **NARAYAN, Shankar
Palo Alto, CA 94306 (US)**

(74) Representative: **Hughes, Andrea Michelle
Frank B. Dehn & Co.,
European Patent Attorneys,
179 Queen Victoria Street
London EC4V 4EL (GB)**

(56) References cited:
**EP-A- 0 140 777        EP-A- 0 232 456
EP-A- 0 515 709**

EP 0 680 652 B1

**Description**

[0001] The present invention relates to systems for smoothly concatenating quasi-periodic waveforms, such as encoded diphone records used in translating text in a computer system to synthesized speech.

[0002] In text-to-speech systems, stored text in a computer is translated to synthesized speech. As can be appreciated, this kind of system would have wide spread application if it were of reasonable cost. For instance, a text-to-speech system could be used for reviewing electronic mail remotely across a telephone line, by causing the computer storing the electronic mail to synthesize speech representing the electronic mail. Also, such systems could be used for reading to people who are visually impaired. In the word processing context, text-to-speech systems might be used to assist in proofreading a large document.

[0003] However in prior art systems which have reasonable cost, the quality of the speech has been relatively poor making it uncomfortable to use or difficult to understand. In order to achieve good quality speech, prior art speech synthesis systems need specialized hardware which is very expensive, and/or a large amount of memory space in the computer system generating the sound.

[0004] In text-to-speech systems, an algorithm reviews an input text string, and translates the words in the text string into a sequence of diphones which must be translated into synthesized speech. Also, text-to-speech systems analyze the text based on word type and context to generate intonation control used for adjusting the duration of the sounds and the pitch of the sounds involved in the speech.

[0005] Diphones consist of a unit of speech composed of the transition between one sound, or phoneme, and an adjacent sound, or phoneme. Diphones typically start at the center of one phoneme and end at the center of a neighboring phoneme. This preserves the transition between the sounds relatively well.

[0006] American English based text-to-speech systems, depending on the particular implementation, use about fifty different sounds referred to as phones. Of these fifty different sounds, the standard language uses about 1800 diphones out of possible 2500 phone pairs. Thus, a text-to-speech system must be capable of reproducing 1800 diphones. To store the speech data directly for each diphone would involve a huge amount of memory. Thus, compression techniques have evolved to limit the amount of memory required for storing the diphones.

[0007] Prior art text-to-speech systems are described in part in United States Patent No. 8,452,168, entitled COMPRESSION OF STORED WAVE FORMS FOR ARTIFICIAL SPEECH, invented by Sprague; and United States Patent No. 4,692,941, entitled REAL-TIME TEXT-TO-SPEECH CONVERSION SYSTEM, invented by Jacks, et al. Further background concerning speech synthesis may be found in United States Patent No. 4,384,169, entitled METHOD AND APPARATUS FOR SPEECH SYNTHESIZING, invented by Mozer, et al.

[0008] Two concatenated diphones will have an ending frame and a beginning frame. The ending frame of the left diphone must be blended with the beginning frame of the right diphone without audible discontinuities or clicks being generated. Since the right boundary of the first diphone and the left boundary of the second diphone correspond to the same phoneme in most situations, they are expected to be similar looking at the point of concatenation. However, because the two diphone codings are extracted from different contexts, they will not look identical. Thus, blending techniques of the prior art have attempted to blend concatenated waveforms at the end and beginning of left and right frames, respectively. Because the end and beginning of frames may not match well, blending noise results. Continuity of sound between adjacent diphones is thus distorted.

[0009] Notwithstanding the prior work in this area, the use of text-to-speech systems has not gained widespread acceptance. It is desireable therefore to provide a software only text-to-speech system which is portable to a wide variety of microcomputer platforms, produces high quality speech and operates in real time on such platforms.

[0010] According to a first aspect, the present invention provides an apparatus for concatenating a first digital frame of N samples having respective magnitudes representing a first quasi-periodic waveform and a second digital frame of M samples having respective magnitudes representing a second quasi-periodic waveform, comprising:

a buffer to store the samples of first and second digital frames;
means, coupled to the buffer store, for determining a blend point for the first and second digital frames in response to magnitudes of samples in the first and second digital frames;
blending means, coupled with the buffer store and the means for determining, for computing a digital sequence representing a concatenation of the first and second quasi-periodic waveforms in response to the first frame, the second frame and the blend point.

[0011] Further, the technique is applicable to concatenating any two quasi-periodic waveforms, commonly encountered in sound synthesis or speech, music, sound effects, or the like.

[0012] According to one embodiment of the present invention, the system operates by first computing an extended frame in response to the first digital frame, and then finding a subset of the extended frame which matches the second digital frame relatively well. The optimum blend point is then defined as a sample in the subset of the extended frame.

The subset of the extended frame which matches the second digital frame relatively well is determined using a minimum average magnitude difference function over the samples in the subset. The blend point in this aspect comprises the first sample of the subset. To generate the concatenated waveform, the subset of the extended frame is combined with the second digital frame and concatenated with the beginning segment of the extended frame to produce the concatenated waveform.

[0013]  The concatenated sequence is then converted to analog form, or other physical representation of the waveforms being blended.

[0014]  According to another aspect, the present invention provides an apparatus for synthesizing speech in response to a text, comprising:

means for translating text to a sequence of sound segment codes;

means, responsive to sound segment codes in the sequence, for decoding the sequence of sound segment codes to produce strings of digital frames of a plurality of samples representing sounds for respective sound segment codes in the sequence, wherein the identified strings of digital frames have beginnings and endings;

means for concatenating a first digital frame at the ending of an identified string of digital frames of a particular sound segment code in the sequences with a second digital frame at the beginning an identified string of digital frames of an adjacent sound segment code in the sequence to produce a speech data sequence, including

a buffer store to store the samples of first and second digital frames;

means, coupled to the buffer store, for determining a blend point for the first and second digital frames in response to magnitudes of samples in the first and second digital frames; and

blending means, coupled with the buffer store and the means for determining, for computing a digital sequence representing a concatenation of the first and second sound segments in response to the first frame, the second frame and the blend point; and

an audio transducer, coupled to the means for concatenating, to generate synthesized speech in response to the speech data sequence.

[0015]  In one embodiment of the invention, the resources that determine the optimum blend point include computing resources that compute an extended frame comprising a discontinuity smoothed concatenation of the first digital frame with a replica of the first digital frame. Further resources find a subset of the extended frame with a minimum average magnitude difference between the samples in the subset and in the second digital frame and define the optimum blend point as the first sample in the subset. The blending resources include software or other computing resources that supply a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence. Next, the second digital frame is combined with the subset of the extended frame, with emphasis on the subset of the extended frame in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence. The first segment and second segment are combined produce the speech data sequence.

[0016]  According to yet further preferred embodiments of the present invention, the text-to-speech apparatus includes a processing module for adjusting the pitch and duration of the identified strings of digital frames in the speech data sequence in response to the input text. Also, the decoder is based on a vector quantization technique which provides excellent quality compression with very small decoding resources required.

[0017]  Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, of preferred embodiments, given by way of example only, and the claims which follow.

[0018]  Fig. 1 is a block diagram of a generic hardware platform incorporating a text-to-speech system according to the present invention.

[0019]  Fig. 2 is a flow chart illustrating a basic text-to-speech routine according to the present invention.

[0020]  Fig. 3 illustrates the format of diphone records according to one embodiment of the present invention.

[0021]  Fig. 4 is a flow chart illustrating an encoder for speech data according to the present invention.

[0022]  Fig. 5 is a graph discussed in reference to the estimation of pitch filter parameters in the encoder of Fig. 4.

[0023]  Fig. 6 is a flow chart illustrating the full search used in the encoder of Fig. 4.

[0024]  Fig. 7 is a flow chart illustrating a decoder for speech data according to the present invention.

[0025]  Fig. 8 is a flow chart illustrating a technique for blending the beginning and ending of adjacent diphone records.

[0026]  Fig. 9 consists of a set of graphs referred to in explanation of the blending technique of Fig. 8.

[0027]  Fig. 10 is a graph illustrating a typical pitch versus time diagram for a sequence of frames of speech data.

[0028]  Fig. 11 is a flow chart illustrating a technique for increasing the pitch period of a particular frame.

[0029]  Fig. 12 is a set of graphs referred to in explanation of the technique of Fig. 11.

[0030]  Fig. 13 is a flow chart illustrating a technique for decreasing the pitch period of a particular frame.

[0031]  Fig. 14 is a set of graphs referred to in explanation of the technique of Fig. 13.

[0032]  Fig. 15 is a flow chart illustrating a technique for inserting a pitch period between two frames in a sequence.

[0033] Fig. 16 is a set of graphs referred to in explanation of the technique of Fig. 15.

[0034] Fig. 17 is a flow chart illustrating a technique for deleting a pitch period in a sequence of frames.

[0035] Fig. 18 is a set of graphs referred to in explanation of the technique of Fig. 17.

[0036] A detailed description of preferred embodiments of the present invention is provided with reference to the figures. Figs. 1 and 2 provide an overview of a system incorporating the present invention. Fig. 3 illustrates the basic manner in which diphone records are stored according to the present invention. Figs. 4-6 illustrate encoding methods based on vector quantization of the present invention. Fig. 7 illustrates a decoding algorithm according to the present invention.

[0037] Figs. 8 and 9 illustrate a preferred technique for blending the beginning and ending of adjacent diphone records. Figs. 10-18 illustrate the techniques for controlling the pitch and duration of sounds in the text-to-speech system.

I. System Overview (Figs. 1-3)

[0038] Fig. 1 illustrates a basic microcomputer platform incorporating a text-to-speech system based on vector quantization according to the present invention. The platform includes a central processing unit 10 coupled to a host system bus 11. A keyboard 12 or other text input device is provided in the system. Also, a display system 13 is coupled to the host system bus. The host system also includes a non-volatile storage system such as a disk drive 14. Further, the system includes host memory 15. The host memory includes text-to-speech (TTS) code, including encoded voice tables, buffers, and other host memory. The text-to-speech code is used to generate speech data for supply to an audio output module 16 which includes a speaker 17. The code also includes an optimum blend point, diphone concatenation routine as described in detail with reference to Figs. 8 and 9.

[0039] According to the present invention, the encoded voice tables include a TTS dictionary which is used to translate text to a string of diphones. Also included is a diphone table which translates the diphones to identified strings of quantization vectors. A quantization vector table is used for decoding the sound segment codes of the diphone table into the speech data for audio output. Also, the system may include a vector quantization table for encoding which is loaded into the host memory 15 when necessary.

[0040] The platform illustrated in Fig. 1 represents any generic microcomputer system, including a Macintosh based system, an DOS based system, a UNIX based system or other types of microcomputers. The text-to-speech code and encoded voice tables according to the present invention for decoding occupy a relatively small amount of host memory 15. For instance, a text-to-speech decoding system according to the present invention may be implemented which occupies less than 640 kilobytes of main memory, and yet produces high quality, natural sounding synthesized speech.

[0041] The basic algorithm executed by the text-to-speech code is illustrated in Fig. 2. The system first receives the input text (block 20). The input text is translated to diphone strings using the TTS dictionary (block 21). At the same time, the input text is analyzed to generate intonation control data, to control the pitch and duration of the diphones making up the speech (block 22).

[0042] After the text has been translated to diphone strings, the diphone strings are decompressed to generate vector quantized data frames (block 23). After the vector quantized (VQ) data frames are produced, the beginnings and endings of adjacent diphones are blended to smooth any discontinuities (block 24). Next, the duration and pitch of the diphone VQ data frames are adjusted in response to the intonation control data (block 25 and 26). Finally, the speech data is supplied to the audio output system for real time speech production (block 27). For systems having sufficient processing power, an adaptive post filter may be applied to further improve the speech quality.

[0043] The TTS dictionary can be implemented using any one of a variety of techniques known in the art. According to the present invention, diphone records are implemented as shown in Fig. 3 in a highly compressed format.

[0044] As shown in Fig. 3, records for a left diphone 30 and a record for a right diphone 31 are shown. The record for the left diphone 30 includes a count 32 of the number NL of pitch periods in the diphone. Next, a pointer 33 is included which points to a table of length NL storing the number $LP_i$ for each pitch period, i goes from 0 to NL-1 of pitch values for corresponding compressed frame records. Finally, pointer 34 is included which points to a table 36 of ML vector quantized compressed speech records, each having a fixed set length of encoded frame size related to nominal pitch of the encoded speech for the left diphone. The nominal pitch is based upon the average number of samples for a given pitch period for the speech data base.

[0045] A similar structure can be seen for the right diphone 31. Using vector quantization, a length of the compressed speech records is very short relative to the quality of the speech generated.

[0046] The format of the vector quantized speech records can be understood further with reference to the frame encoder routine and the frame decoder routine described below with reference to Figs. 4-7.

II. The Encoder/Decoder Routines (Figs. 4-7)

[0047]   The encoder routine is illustrated in Fig. 4. The encoder accepts as input a frame $s_n$ of speech data. In the preferred system, the speech samples are represented as 12 or 16 bit two's complement numbers, sampled at 22,252 Hz. This data is divided into non-overlapping frames $s_n$ having a length of N, where N is referred to as the frame size. The value of N depends on the nominal pitch of the speech data. If the nominal pitch of the recorded speech is less than 165 samples (or 135 Hz), the value of N is chosen to be 96. Otherwise a frame size of 160 is used. The encoder transforms the N-point data sequence $s_n$ into a byte stream of shorter length, which depends on the desired compression rate. For example, if N = 160 and very high data compression is desired, the output byte stream can be as short as 12 eight bit bytes. A block diagram of the encoder is shown in Fig. 4.

[0048]   Thus, the routine begins by accepting a frame $s_n$ (block 50). To remove low frequency noise, such as DC or 60 Hz power line noise, and produce offset free speech data, signal $s_n$ is passed through a high pass filter. A difference equation used in a preferred system to accomplish this is set out in Equation 1 for $0 \leq n < N$.

$$x_n = s_n - s_{n-1} + 0.999 * x_{n-1}$$   Equation 1

[0049]   The value $x_n$ is the "offset free" signal. The variables $s_{-1}$ and $x_{-1}$ are initialized to zero for each diphone and are subsequently updated using the relation of Equation 2.

$$x_{-1} = x_N \text{ and } s_{-1} = s_N$$   Equation 2

[0050]   This step can be referred to as offset compensation or DC removal (block 51).

[0051]   In order to partially decorrelate the speech samples and the quantization noise, the sequence $x_n$ is passed through a fixed first order linear prediction filter. The difference equation to accomplish this is set forth in Equation 3.

$$y_n = x_n - 0.875 * x_{n-1}$$   Equation 3

[0052]   The linear prediction filtering of Equation 3 produces a frame $y_n$ (block 52). The filter parameter, which is equal to 0.875 in Equation 3, will have to be modified if a different speech sampling rate is used. The value of $x_{-1}$ is initialized to zero for each diphone, but will be updated in the step of inverse linear prediction filtering (block 60) as described below.

[0053]   It is possible to use a variety of filter types, including, for instance, an adaptive filter in which the filter parameters are dependent on the diphones to be encoded, or higher order filters.

[0054]   The sequence $y_n$ produced by Equation 3 is then utilized to determine an optimum pitch value, $P_{opt}$ and an associated gain factor, $\beta$. $P_{opt}$ is computed using the functions $s_{xy}(P)$ $s_{xx}(P)$, $s_{yy}(P)$, and the coherence function Coh (P) defined by Equations 4, 5, 6 and 7 as set out below.

$$s_{xy}(P) = \sum_{n=0}^{N-1} y_n * PBUF_{P_{max} - P + n}$$

Equation 4

$$s_{xx}(P) = \sum_{n=0}^{N-1} y_n * y_n$$

Equation 5

$$s_{yy}(P) = \sum_{N=0}^{N-1} PBUF_{P_{max} - P + n} \ast PBUF_{P_{max} - P + n}$$

**Equation 6**

and

$$Coh(P) = s_{xy}(P) \ast s_{xy}(P)/(s_{xx}(P) \ast s_{yy}(P))$$
Equation 7

[0055]   PBUF is a pitch buffer of size $P_{max}$, which is initialized to zero, and updated in the pitch buffer update block 59 as described below. P opt is the value of P for which Coh(P) is maximum and $s_{xy}(P)$ is positive. The range of P considered depends on the nominal pitch of the speech being coded. The range is (96 to 350) if the frame size is equal to 96 and is (160 to 414) if the frame size is equal to 160. $P_{max}$ is 350 if nominal pitch is less than 160 and is equal to 414 otherwise. The parameter $P_{opt}$ can be represented using 8 bits.

[0056]   The computation of $P_{opt}$ can be understood with reference to Fig. 5. In Fig. 5, the buffer PBUF is represented by the sequence 100 and the frame $y_n$ is represented by the sequence 101. In a segment of speech data in which the preceding frames are substantially equal to the frame $y_n$, PBUF and $y_n$ will look as shown in Fig. 5. $P_{opt}$ will have the value at point 102, where the vector $y_n$ 101 matches as closely as possible a corresponding segment of similar length in PBUF 100.

[0057]   The pitch filter gain parameter β is determined using the expression of Equation 8.

$$\beta = s_{xy}(P_{opt}) / s_{yy}(P_{opt}).$$
Equation 8

[0058]   β is quantized to four bits, so that the quantized value of β can range from 1/16 to 1, in steps of 1/16.

[0059]   Next, a pitch filter is applied (block 54). The long term correlations in the pre-emphasized speech data $y_n$ are removed using the relation of Equation 9.

$$r_n = y_n - \beta \ast PBUF_{P_{max} - P_{opy} + n,} \qquad 0 \le n < N.$$
Equation 9

[0060]   This results in computation of a residual signal $r_n$.

[0061]   Next, a scaling parameter G is generated using a block gain estimation routine (block 55). In order to increase the computational accuracy of the following stages of processing, the residual signal $r_n$ is rescaled. The scaling parameter, G, is obtained by first determining the largest magnitude of the signal $r_n$ and quantizing it using a 7-level quantizer. The parameter G can take one of the following 7 values: 256, 512, 1024, 2048, 4096, 8192, and 16384. The consequence of choosing these quantization levels is that the rescaling operation can be implemented using only shift operations.

[0062]   Next the routine proceeds to residual coding using a full search vector quantization code (block 56). In order to code the residual signal $r_n$, the n point sequence $r_n$ is divided into non-overlapping blocks of length M, where M is referred to as the "vector size". Thus, M sample blocks $b_{ij}$ are created, where i is an index from zero to M-1 on the block number, and j is an index from zero to N/M-1 on the sample within the block. Each block may be defined as set out in Equation 10.

$$b_{ij} = r_{Mi+j}, \quad (0 \le i < N/M \text{ and } j \le 0 < M)$$
Equation 10

[0063]   Each of these M sample blocks $b_{ij}$ will be coded into an 8 bit number using vector quantization. The value of M depends on the desired compression ratio. For example, with M equal to 16, very high compression is achieved (i. e., 16 residual samples are coded using only 8 bits). However, the decoded speech quality can be perceived to be somewhat noisy with M=16. On the other hand, with M=2, the decompressed speech quality will be very close to that of uncompressed speech. However the length of the compressed speech records will be longer. The preferred imple-

mentation, the value M can take values 2, 4, 8, and 16.

[0064] The vector quantization is performed as shown in Fig. 6. Thus, for all blocks $b_{ij}$ a sequence of quantization vectors is identified (block 120). First, the components of block $b_{ij}$ are passed through a noise shaping filter and scaled as set out in Equation 11 (block 121).

$$w_j = 0.875 * w_{j-1} - 0.5 * w_{j-2} + 0.4375 * w_{j-3} + b_{ij},$$

$$0 \leq j < M$$

$$v_{ij} = G * w_j \qquad 0 \leq j < M \qquad \text{Equation 11}$$

[0065] Thus, $v_{ij}$ is the jth component of the vector $v_i$, and the values $w_{-1}$, $w_{-2}$ and $w_{-3}$ are the states of the noise shaping filter and are initialized to zero for each diphone. The filter coefficients are chosen to shape the quantization noise spectra in order to improve the subjective quality of the decompressed speech. After each vector is coded and decoded, these states are updated as described below with reference to blocks 124-126.

[0066] Next, the routine finds a pointer to the best match in a vector quantization table (block 122). The vector quantization table 123 consists of a sequence of vectors $C_0$ through $C_{255}$ (block 123).

[0067] Thus, the vector $v_i$ is compared against 256 M-point vectors, which are precomputed and stored in the code table 123. The vector $C_{qi}$ which is closest to $v_i$ is determined according to Equation 12. The value $C_p$ for p = 0 through 255 represents the $p^{th}$ encoding vector from the vector quantization code table 123.

$$\min_p \sum_{j=0}^{M-1} (v_{ij} - C_{pj})^2$$

**Equation 12**

[0068] The closest vector $C_{qi}$ can also be determined efficiently using the technique of Equation 13.

$$v_i^T \bullet C_{qi} \leq v_i^T \bullet C_p \text{ for all } p(0 \leq p \leq 255) \qquad \text{Equation 13}$$

In Equation 13, the value vT represents the transpose of the vector v, and "•" represents the inner product operation in the inequality.

[0069] The encoding vectors $C_p$ in table 123 are utilized to match on the noise filtered value $v_{ij}$. However in decoding, a decoding vector table 125 is used which consists of a sequence of vectors $QV_p$. The values $QV_p$ are selected for the purpose of achieving quality sound data using the vector quantization technique. Thus, after finding the vector $C_{qi}$, the pointer q is utilized to access the vector $QV_{qi}$. The decoded samples corresponding to the vector $b_i$ which is produced at step 55 of Fig. 4, is the M-point vector $(1/G) * QV_{qi}$. The vector $C_p$ is related to the vector $QV_p$ by the noise shaping filter operation of Equation 11. Thus, when the decoding vector $QV_p$ is accessed, no inverse noise shaping filter needs to be computed in the decode operation. The table 125 of Fig. 6 thus includes noise compensated quantization vectors.

[0070] In continuing to compute the encoding vectors for the vectors $b_{ij}$ which make up the residual signal $r_n$, the decoding vector of the pointer to the vector $b_i$ is accessed (block 124). That decoding vector is used for filter and PBUF updates (block 126).

[0071] For the noise shaping filter, after the decoded samples are computed for each sub-block $b_i$, the error vector $(b_i - QV_{qi})$ is passed through the noise shaping filter as shown in Equation 14.

$$W_j = 0.875 * W_{j-1} - 0.5 * W_{j-2} + 0.4375 * W_{j-3} + [b_{ij} - QV_{qi}(j)]$$

$$0 \leq j < M \qquad \text{Equation 14}$$

[0072] In Equation 14, the value $QV_{qi}(j)$ represents the $j^{th}$ component of the decoding vector $QV_{qi}$. The noise shaping filter states for the next block are updated as shown in Equation 15.

$$w_{-1} = w_{M-1}$$

$$w_{-2} = w_{M-2}$$

$$w_{-3} = w_{M-3}$$

Equation 15

[0073]    This coding and decoding is performed for all of the N/M sub-blocks to obtain N/M indices to the decoding vector table 125. This string of indices $Q_n$, for n going from zero to N/M-1 represent identifiers for a string of decoding vectors for the residual signal $r_n$.

[0074]    Thus, four parameters represent the N-point data sequence $y_n$:

1) Optimum pitch, $P_{opt}$ (8 bits),
2) Pitch filter gain, $\beta$ (4 bits),
3) Scaling parameter, G (3 bits), and
4) A string of decoding table indices, $Q_n$ ($0 \leq n < $ N/M).

[0075]    The parameters $\beta$ and G can be coded into a single byte. Thus, only (N/M) plus 2 bytes are used to represent N samples of speech. For example, suppose nominal pitch is 100 samples long, and M = 16. In this case, a frame of 96 samples of speech are represented by 8 bytes: 1 byte for $P_{opt}$, 1 byte for $\beta$ and G, and 6 bytes for the decoding table indices $Q_n$. If the uncompressed speech consists of 16 bit samples, then this represents a compression of 24:1.

[0076]    Back to Fig. 4, four parameters identifying the speech data are stored (block 57). In a preferred system, they are stored in a structure as described with respect to Fig. 3 where the structure of the frame can be characterized as follows:

```
#define         NumOfVectorsPerFrame   (FrameSize / VectorSize)


struct frame {
        unsigned    Gain : 4;
        unsigned    Beta : 3;
        unsigned    UnusedBit: 1;
        unsigned    char Pitch ;
        unsigned    char VQcodes[NumOfVectorsPerFrame]; };
```

[0077]    The diphone record of Fig. 3 utilizing this frame structure can be characterized as follows:

```
DiphoneRecord
{
        char   LeftPhone, RightPhone;
        short  LeftPitchPeriodCount,RightPitchPeriodCount;
        short  *LeftPeriods, *RightPeriods;
        struct      frame *LeftData, *RightData;
}
```

[0078]    These stored parameters uniquely provide for identification of the diphones required for text-to-speech synthesis.

[0079]    As mentioned above with respect to Fig. 6, the encoder continues decoding the data being encoded in order to update the filter and PBUF values. The first step involved in this is an inverse pitch filter (block 58). With the vector $r'_n$ corresponding to the decoded signal formed by concatenating the string of decoding vectors to represent the residual

signal $r'_n$, the inverse filter is implemented as set out in Equation 16.

$$y'_n = r'_n + \beta * PBUF_{Pmax - Popt + n'} \qquad 0 \leq n < N. \qquad \text{Equation 16}$$

[0080]   Next, the pitch buffer is updated (block 59) with the output of the inverse pitch filter. The pitch buffer PBUF is updated as set out in Equation 17.

$$PBUF_n = PBUF_{(n + N)} \qquad 0 \leq n < (P_{max} - N)$$

$$PBUF_{(Pmax - N + n)} = y'_n \qquad 0 \leq n < N \qquad \text{Equation 17}$$

[0081]   Finally, the linear prediction filter parameters are updated using an inverse linear prediction filter step (block 60). The output of the inverse pitch filter is passed through a first order inverse linear prediction filter to obtain the decoded speech. The difference equation to implement this filter is set out in Equation 18.

$$x'_n = 0.875 * x'_{n-1} + y'_n \qquad \text{Equation 18}$$

[0082]   In Equation 18, $x'_n$ is the decompressed speech. From this, the value of $x_{-1}$ for the next frame is set to the value $x_N$ for use in the step of block 52.

[0083]   Fig. 7 illustrates the decoder routine. The decoder module accepts as input (N/M) + 2 bytes of data, generated by the encoder module, and applies as output N samples of speech. The value of N depends on the nominal pitch of the speech data and the value of M depends on the desired compression ratio.

[0084]   In software only text-to-speech systems, the computational complexity of the decoder must be as small as possible to ensure that the text-to-speech system can run in real time even on slow computers. A block diagram of the decoder is shown in Fig. 7.

[0085]   The routine starts by accepting diphone records at block 200. The first step involves parsing the parameters G, $\beta$, $P_{opt}$, and the vector quantization string $Q_n$ (block 201). Next, the residual signal $r'_n$ is decoded (block 202). This involves accessing and concatenating the decoding vectors for the vector quantization string as shown schematically at block 203 with access to the decoding vector table 125.

[0086]   After the residual signal $r'_n$ is decoded, an inverse pitch filter is applied (block 204). This inverse pitch filter is implemented as shown in Equation 19:

$$y'_n = r'_n + \beta * SPBUF(P_{max} - P_{opt} + n), 0 \leq n < N. \qquad \text{Equation 19}$$

SPBUF is a synthesizer pitch buffer of length $P_{max}$ initialized as zero for each diphone, as described above with respect to the encoder pitch buffer PBUF.

[0087]   For each frame, the synthesis pitch buffer is updated (block 205). The manner in which it is updated is shown in Equation 20:

$$SPBUF_n = SPBUF_{(n + N)} \qquad 0 \leq n < (P_{max} - N)$$

$$SPBUF_{(Pmax - N + n)} = y'_n \qquad 0 \leq n < N \qquad \text{Equation 20}$$

[0088]   After updating SPBUF, the sequence $y'_n$ is applied to an inverse linear prediction filtering step (block 206). Thus, the output of the inverse pitch filter $y'_n$ is passed through a first order inverse linear prediction filter to obtain the decoded speech. The difference equation to implement the inverse linear prediction filter is set out in Equation 21:

$$x'_n = 0.875 * x'_{n-1} + y'_n \qquad \text{Equation 21}$$

[0089]   In Equation 21, the vector $x'_n$ corresponds to the decompressed speech. This filtering operation can be implemented using simple shift operations without requiring any multiplication. Therefore, it executes very quickly and

utilizes a very small amount of the host computer resources.

[0090] Encoding and decoding speech according to the algorithms described above, provide several advantages over prior art systems. First, this technique offers higher speech compression rates with decoders simple enough to be used in the implementation of software only text-to-speech systems on computer systems with low processing power. Second, the technique offers a very flexible trade-off between the compression ratio and synthesizer speech quality. A high-end computer system can opt for higher quality synthesized speech at the expense of a bigger RAM memory requirement.

## III. Waveform Blending For Discontinuity Smoothing (Figs. 8 and 9)

[0091] As mentioned above with respect to Fig. 2, the synthesized frames of speech data generated using the vector quantization technique may result in slight discontinuities between diphones in a text string. Thus, the text-to-speech system provides a module for blending the diphone data frames to smooth such discontinuities. The blending technique of the preferred embodiment is shown with respect to Figs. 8 and 9.

[0092] Two concatenated diphones will have an ending frame and a beginning frame. The ending frame of the left diphone must be blended with the beginning frame of the right diphone without audible discontinuities or clicks being generated. Since the right boundary of the first diphone and the left boundary of the second diphone correspond to the same phoneme in most situations, they are expected to be similar looking at the point of concatenation. However, because the two diphone codings are extracted from different context, they will not look identical. This blending technique is applied to eliminate discontinuities at the point of concatenation. In Fig. 9, the last frame, referring here to one pitch period, of the left diphone is designated $L_n$ ($0 \leq n < PL$) at the top of the page. The first frame (pitch period) of the right diphone is designated $R_n$ ($0 \leq n < PR$). The blending of $L_n$ and $R_n$ according to the present invention will alter these two pitch periods only and is performed as discussed with reference to Fig. 8. The waveforms in Fig. 9 are chosen to illustrate the algorithm, and may not be representative of real speech data.

[0093] Thus, the algorithm as shown in Fig. 8 begins with receiving the left and right diphone in a sequence (block 300). Next, the last frame of the left diphone is stored in the buffer $L_n$ (block 301). Also, the first frame of the right diphone is stored in buffer $R_n$ (block 302).

[0094] Next, the algorithm replicates and concatenates the left frame $L_n$ to form extend frame (block 303). In the next step, the discontinuities in the extended frame between the replicated left frames are smoothed (block 304). This smoothed and extended left frame is referred to as $EI_n$ in Fig. 9.

[0095] The extended sequence $EI_n$ ($0 \leq n < 2*PL$) is obtained in the first step as shown in Equation 22:

$$EI_n = L_n \qquad n = 0,1,...,PL\text{-}1$$

$$EI_{PL+n} = L_n \qquad n = 0,1,...,PL\text{-}1 \qquad \text{Equation 22}$$

Then discontinuity smoothing from the point n = PL is conducted according to the filter of Equation 23:

$$EI_{PL+n} = EI_{PL+n} + [EI_{(PL\text{-}1)} - EI'_{(PL\text{-}1)}]^* \Delta^{n+1},$$

$$n = 0,1,...,(PL/2). \qquad \text{Equation 23}$$

In Equation 23, the value $\Delta$ is equal to 15/16 and $EI'_{(PL\text{-}1)} = EI_2 + 3 * (EI_1\text{-}EI_0)$. Thus, as indicated in Fig. 9, the extended sequence $EI_n$ is substantially equal to $L_n$ on the left hand side, has a smoothed region beginning at the point PL and converges on the original shape of $L_n$ toward the point 2PL. If $L_n$ was perfectly periodic, then $EI_{PL\text{-}1} = EI'_{PL\text{-}1}$.

[0096] In the next step, the optimum match of $R_n$ with the vector $EI_n$ is found. This match point is referred to as $P_{opt}$. (Block 305.) This is accomplished essentially as shown in Fig. 9 by comparing $R_n$ with $EI_n$ to find the section of $EI_n$ which most closely matches $R_n$. This optimum blend point determination is performed using Equation 24 where W is the minimum of PL and PR, and AMDF represents the average magnitude difference function.

$$\text{AMDF}(p) = \sum_{n=0}^{W-1} | EI_{n+p} - R_n |$$

**Equation 24**

[0097] This function is computed for values of p in the range of 0 to PL-1. The vertical bars in the operation denote the absolute value. W is the window size for the AMDF computation. $P_{opt}$ is chosen to be the value at which AMDF(p) is minimum. This means that $p = P_{opt}$ corresponds to the point at which sequences $EI_{n+p}$ ($0 \leq n < W$) and $R_n$ ($0 \leq n < W$) are very close to each other.

[0098] After determining the optimum blend point $P_{opt}$, the waveforms are blended (block 306). The blending utilizes a first weighting ramp WL which is shown in Fig. 9 beginning at $P_{opt}$ in the $EI_n$ trace. In a second ramp, WR is shown in Fig. 9 at the $R_n$ trace which is lined up with $P_{opt}$. Thus, in the beginning of the blending operation, the value of $EI_n$ is emphasized. At the end of the blending operation, the value of $R_n$ is emphasized.

[0099] Before blending, the length PL of $L_n$ is altered as needed to ensure that when the modified $L_n$ and $R_n$ are concatenated, the waveforms are as continuous as possible. Thus, the length P'L is set to $P_{opt}$ if $P_{opt}$ is greater than PL/2. Otherwise, the length P'L is equal to $W + P_{opt}$ and the sequence $L_n$ is equal to $EI_n$ for $0 \leq n \leq$ (P'L-1).

[0100] The blending ramp beginning at $P_{opt}$ is set out in Equation 25:

$$R_n = EI_{n+Popt} + (R_n - EI_{n+Popt})^*(n+1)/W \qquad 0 \leq n < W$$

$$R_n = R_n \qquad W \leq n < PR$$

**Equation 25**

[0101] Thus, the sequences $L_n$ and $R_n$ are windowed and added to get the blended $R_n$. The beginning of $L_n$ and the ending of $R_n$ are preserved to prevent any discontinuities with adjacent frames.

[0102] This blending technique is believed to minimize blending noise in synthesized speech produced by any concatenated speech synthesis.

IV. Pitch and Duration Modification (Figs. 10-18)

[0103] As mentioned above with respect to Fig. 2, a text analysis program analyzes the text and determines the duration and pitch contour of each phone that needs to be synthesized and generates intonation control signals. A typical control for a phone will indicate that a given phoneme, such as AE, should have a duration of 200 milliseconds and a pitch should rise linearly from 220Hz to 300Hz. This requirement is graphically shown in Fig. 10. As shown in Fig. 10, T equals the desired duration (e.g. 200 milliseconds) of the phoneme. The frequency $f_b$ is the desired beginning pitch in Hz. The frequency $f_e$ is the desired ending pitch in Hz. The labels $P_1$, $P_2$...,$P_6$ indicate the number of samples of each frame to achieve the desired pitch frequencies $f_b$, $f_2$...,$f_6$. The relationship between the desired number of samples, $P_i$, and the desired pitch frequency $f_i$ ($f_1 = f_b$), is defined by the relation:

$P_i = F_s/f_i$, where $F_s$ is the sampling frequency for the data.

As can be seen in Fig. 10, the pitch period for a lower frequency period of the phoneme is longer than the pitch period for a higher frequency period of the phoneme. If the nominal frequency were $P_3$, then the algorithm would be required to lengthen the pitch period for frames $P_1$ and $P_2$ and decrease the pitch periods for frames $P_4$, $P_5$ and $P_6$. Also, the given duration T of the phoneme will indicate how many pitch periods should be inserted or deleted from the encoded phoneme to achieve the desired duration period. Figs. 11 through 18 illustrate a preferred implementation of such algorithms.

[0104] Fig. 11 illustrates an algorithm for increasing the pitch period, with reference to the graphs of Fig. 12. The algorithm begins by receiving a control to increase the pitch period to N+Δ, where N is the pitch period of the encoded frame. (Block 350). In the next step, the pitch period data is stored in a buffer $x_n$ (block 351). $x_n$ is shown in Fig. 12 at the top of the page. In the next step, a left vector $L_n$ is generated by applying a weighting function WL to the pitch period data $x_n$ with reference to Δ (block 352). This weighting function is illustrated in Equation 26 where M = N-Δ:

$$L_n = x_n \qquad \text{for } 0 \leq n < \Delta$$

$$L_n = x_n * (N-n)/(M+1) \qquad \text{for } \Delta \le n < N \qquad\qquad \text{Equation 26}$$

As can be seen in Fig. 12, the weighting function WL is constant from the first sample to sample $\Delta$, and decreases from $\Delta$ to N.

[0105]  Next, a weighting function WR is applied to $x_n$ (block 353) as can be seen in the Fig. 12. This weighting function is executed as shown in Equation 27:

$$R_n = x_{n+\Delta} *(n+1)/(M+1) \qquad \text{for } 0 \le n < N-\Delta$$

$$R_n = X_{n+\Delta} \qquad \text{for } N-\Delta \le n < N. \qquad\qquad \text{Equation 27}$$

[0106]  As can be seen in Fig. 12, the weighting function WR increases from 0 to N-$\Delta$ and remains constant from N-$\Delta$ to N. The resulting waveforms $L_n$ and $R_n$ are shown conceptually in Fig. 12. As can be seen, $L_n$ maintains the beginning of the sequence $x_n$, while $R_n$ maintains the ending of the data $x_n$.

[0107]  The pitch modified sequence $y_n$ is formed (block 354) by adding the two sequences as shown in Equation 28:

$$y_n = L_n + R_{(n-\Delta)} \qquad\qquad \text{Equation 28}$$

This is graphically shown in Fig. 12 by placing $R_n$ shifted by $\Delta$ below $L_n$. The combination of $L_n$ and $R_n$ shifted by $\Delta$ is shown to be $y_n$ at the bottom of Fig. 12. The pitch period for $y_n$ is N+$\Delta$. The beginning of $y_n$ is the same as the beginning of $x_n$, and the ending of $y_n$ is substantially the same as the ending of $x_n$. This maintains continuity with adjacent frames in the sequence, and accomplishes a smooth transition while extending the pitch period of the data.

[0108]  Equation 28 is executed with the assumption that $L_n$ is 0, for $n \le N$, and $R_n$ is 0 for $n < 0$. This is illustrated pictorially in Fig. 12.

[0109]  An efficient implementation of this scheme which requires at most one multiply per sample, is shown in Equation 29:

$$y_n = x_n \qquad 0 \le n < \Delta$$

$$y_n = x_n + [x_{n-\Delta} - x_n]*(n-\Delta+1)/(N-\Delta+1) \qquad \Delta \le n < N$$

$$y_n = x_{n-\Delta}$$

$$N \le n < N_d \qquad\qquad \text{Equation 29}$$

This results in a new pitch period having a pitch period of N+$\Delta$.

[0110]  There are also instances in which the pitch period must be decreased. The algorithm for decreasing the pitch period is shown in Fig. 13 with reference to the graphs of Fig. 14. Thus, the algorithm begins with a control signal indicating that the pitch period must be decreased to N-$\Delta$. (Block 400). The first step is to store two consecutive pitch periods in the buffer $x_n$ (block 401). Thus, the buffer $x_n$ as can be seen in Fig. 14 consists of two consecutive pitch periods, with the period $N_l$ being the length of the first pitch period, and $N_r$ being the length of the second pitch period. Next, two sequences $L_n$ and $R_n$ are conceptually created using weighting functions WL and WR (blocks 402 and 403). The weighting function WL emphasizes the beginning of the first pitch period, and the weighting function WR emphasizes the ending of the second pitch period. These functions can be conceptually represented as shown in Equations 30 and 31, respectively:

$$L_n = x_n \qquad \text{for } 0 \le n < N_l - W$$

$$L_n = x_n * (N_l - n)/(W+1) \qquad W \le n < N_l$$

$$L_n = 0 \qquad \text{otherwise.} \qquad\qquad \text{Equation 30}$$

and

$$R_n = x_n * (n-N_l+W-\Delta+1)/(W+1) \qquad \text{for } N_l-W+\Delta \leq n < N_l+\Delta$$

$$R = x_n \qquad \text{for } N_l+\Delta \leq n < N_l+N_r$$

$$R_n = 0 \qquad \text{otherwise.} \qquad\qquad \text{Equation 31}$$

[0111]   In these equations, $\Delta$ is equal to the difference between $N_l$ and the desired pitch period $N_d$. The value W is equal to $2^*\Delta$, unless $2^*\Delta$ is greater than $N_d$, in which case W is equal to $N_d$.

[0112]   These two sequences $L_n$ and $R_n$ are blended to form a pitch modified sequence $y_n$ (block 404). The length of the pitch modified sequence $y_n$ will be equal to the sum of the desired length and the length of the right phoneme frame $N_r$. It is formed by adding the two sequences as shown in Equation 32:

$$y_n = L_n + R_{(n+\Delta)} \qquad\qquad \text{Equation 32}$$

[0113]   Thus, when a pitch period is decreased, two consecutive pitch periods of data are affected, even though only the length of one pitch period is changed. This is done because pitch periods are divided at places where short-term energy is the lowest within a pitch period. Thus, this strategy affects only the low energy portion of the pitch periods. This minimizes the degradation in speech quality due to the pitch modification. It should be appreciated that the drawings in Fig. 14 are simplified and do not represent actual pitch period data.

[0114]   An efficient implementation of this scheme, which requires at most one multiply per sample, is set out in Equations 33 and 34.

[0115]   The first pitch period of length $N_d$ is given by Equation 33:

$$y_n = x_n \qquad 0 \leq n < N_l-W$$

$$y_n = x_n + [x_{n+\Delta} - x_n]^*(n-N_l+W+1)/(W+1) \qquad N_l-W \leq n < N_d \qquad \text{Equation 33}$$

[0116]   The second pitch period of length $N_r$ is generated as shown in Equation 34:

$$y_n = x_{n-\Delta} + [x_n - x_{n-\Delta}]^*(n-\Delta-N_l+W+1)/(W+1)$$

$$N_l \leq n < N_l+\Delta$$

$$y_n = x_n \qquad N_{l+\Delta} \leq n < N_l+N_r \qquad\qquad \text{Equation 34}$$

[0117]   As can be seen in Fig. 14, the sequence $L_n$ is essentially equal to the first pitch period until the point $N_l-W$. At that point, a decreasing ramp WL is applied to the signal to dampen the effect of the first pitch period.

[0118]   As also can be seen, the weighting function WR begins at the point $N_l-W + \Delta$ and applies an increasing ramp to the sequence $x_n$ until the point $N_l + \Delta$. From that point, a constant value is applied. This has the effect of damping the effect of the right sequence and emphasizing the left during the beginning of the weighting functions, and generating an ending segment which is substantially equal to the ending segment of $x_n$ emphasizing the right sequence and damping the left. When the two functions are blended, the resulting waveform $y_n$ is substantially equal to the beginning of $x_n$ at the beginning of the sequence, at the point $N_l-W$ a modified sequence is generated until the point $N_l$. From $N_l$ to the ending, sequence $x_n$ is shifted by $\Delta$ results.

[0119]   A need also arises for insertion of pitch periods to increase the duration of a given sound. A pitch period is inserted according to the algorithm shown in Fig. 15 with reference to the drawings of Fig. 16.

[0120]   The algorithm begins by receiving a control signal to insert a pitch period between frames $L_n$ and $R_n$ (block 450). Next, both $L_n$ and $R_n$ are stored in the buffer (block 451), where $L_n$ and $R_n$ are two adjacent pitch periods of a voice diphone. (Without loss of generality, it is assumed for the description that the two sequences are of equal lengths N.)

[0121]   In order to insert a pitch period, $x_n$ of the same duration, without causing a discontinuity between $L_n$ and $x_n$

and between $x_n$ and $R_n$, the pitch period $x_n$ should resemble $R_n$ around n=0 (preserving $L_n$ to $x_n$ continuity), and should resemble $L_n$ around n=N (preserving $x_n$ to $R_n$ continuity). This is accomplished by defining $x_n$ as shown in Equation 35:

$$x_n = R_n + (L_n - R_n) * [(n+1)/(N+1)] \qquad 0 \leq n < N-1 \qquad \text{Equation 35}$$

[0122] Conceptually, as shown in Fig. 15, the algorithm proceeds by generating a left vector WL($L_n$), essentially applying to the increasing ramp WL to the signal $L_n$. (Block 452).

[0123] A right vector WR ($R_n$) is generated using the weighting vector WR (block 453) which is essentially a decreasing ramp as shown in Fig. 16. Thus, the ending of $L_n$ is emphasized with the left vector, and the beginning of $R_n$ is emphasized with the vector WR.

[0124] Next, WR ($L_n$) and WR ($R_n$) are blended to create an inserted period $x_n$ (block 454).

[0125] The computation requirement for inserting a pitch period is thus just a multiplication and two additions per speech sample.

[0126] Finally, concatenation of $L_n$, $x_n$ and $R_n$ produces a sequence with an inserted pitch period (block 455).

[0127] Deletion of a pitch period is accomplished as shown in Fig. 17 with reference to the graphs of Fig. 18. This algorithm, which is very similar to the algorithm for inserting a pitch period, begins with receiving a control signal indicating deletion of pitch period $R_n$ which follows $L_n$ (block 500). Next, the pitch periods $L_n$ and $R_n$ are stored in the buffer (block 501). This is pictorially illustrated in Fig. 18 at the top of the page. Again, without loss of generality, it is assumed that the two sequences have equal lengths N.

[0128] The algorithm operates to modify the pitch period $L_n$ which precedes $R_n$ (to be deleted) so that it resembles $R_n$, as n approaches N. This is done as set forth in Equation 36:

$$L'_n = L_n + (R_n - L_n) * [(n+1)/(N+1)] \qquad 0 \leq n < N-1 \qquad \text{Equation 36}$$

In Equation 36, the resulting sequence $L'_n$ is shown at the bottom of Fig. 18. Conceptually, Equation 36 applies a weighting function WL to the sequence $L_n$ (block 502). This emphasizes the beginning of the sequence $L_n$ as shown. Next, a right vector WR ($R_n$) is generated by applying a weighting vector WR to the sequence $R_n$ that emphasizes the ending of $R_n$ (block 503).

[0129] WL ($L_n$) and WR ($R_n$) are blended to create the resulting vector $L'_n$. (Block 504). Finally, the sequence $L_n$-$R_n$ is replaced with the sequence $L'_n$ in the pitch period string. (Block 505).

IV. Conclusion

[0130] Accordingly, the present invention presents a software only text-to-speech system which is efficient, uses a very small amount of memory, and is portable to a wide variety of standard microcomputer platforms. It takes advantage of knowledge about speech data, and to create a speech compression, blending, and duration control routine which produces very high quality speech with very little computational resources.

[0131] A source code listing of the software for executing the compression and decompression, the blending, and the duration and pitch control routines is provided in the Appendix as an example of a preferred embodiment of the present invention.

[0132] The foregoing description of preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims.

APPENDIX

5

COMPUTER PROGRAM LISTINGS

[0133]

10

15

20

25

30

35

40

45

50

55

**I. ENCODER MODULE**

[0134]

5

```
#include <stdio.h>
#include <math.h>
#include <StdLib.h>
#include <types.h>
#include <fcntl.h>
#include <string.h>

#include <types.h>
#include <files.h>
#include <resources.h>
#include <memory.h>
#include "vqcoder.h"

#define    LAST_FRAME_FLAG          128
#define    PBUF_SIZE  440
static float        oc_state[2], nsf_state[NSF_ORDER + 1];
static short        pstate[PORDER + 1], dstate[PORDER + 1];
static short        AnaPbuf[PBUF_SIZE];

static short        vsize, cbook_size, bs_size;

#pragma segment vqlib

/* Read Code Books */
float   *EncodeBook[MAX_CBOOK_SIZE];
short   *DecodeBook[MAX_CBOOK_SIZE];
get_cbook(short ratio)
{
    short *p;
    short    frame_size, i;
    static    short last_ratio = 0;

    Handle h;
    int        skip;
    h = GetResource('CBOK',1);
    HLock(h);
    p = (short *) *h;

    if (ratio = = last_ratio)
        return;
    last_ratio = ratio;

    if (ratio < 3)
        return;

    if (NOMINAL_PITCH < 165)
```

10

15

20

25

30

35

40

45

50

55

```
            frame_size = 96;
        else
            frame_size = 160;

        get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);
        skip = 0;
        while (p[skip + 1] != vsize)
        {
            short t1, t2;
            t2 = p[skip];
            t1 = p[skip + 1];
            skip += sizeof(float) * (2 * t2-1) * (t1 + 1) / sizeof(short)
                + (2 * t2 * t1 + 2);
        }

        /*Skip Binary search tree */
        skip += sizeof(float) * (cbook_size-1) * (vsize + 1) / sizeof(short)
            + (cbook_size * vsize + 2);

        /* Get pointers to Full search code books */
        for (i = 0; i < cbook_size; i++)
        {
            EncodeBook[i] = (float *) &p[skip];
            skip += (vsize + 1) * sizeof (float) / sizeof(short);
        }

        for (i = 0; i < cbook_size; i++)
        {
            DecodeBook[i] = p + skip;
            skip += vsize;
        }
    }

char *getcbook(long *len, short ratio)
{
    get_cbook(ratio);
    *len = sizeof(short) * vsize * cbook_size;
        /* plus one is to make space at the end for the array of pointers */
    return (char*) DecodeBook[0];
}

/* A Routine for Pitch filter parameter Estimation */
GetPitchFilterPars (x, len, pbuf, min_pitch, max_pitch, pitch, beta)
float   *beta;
short   *x, *pbuf;
short   min_pitch, max_pitch;
short   len;
unsigned int *pitch;
{
    /* Estimate long-term predictor */
```

```
int     best_pitch, i, j;
float   syy, sxy, best_sxy = 0.0, best_syy = 1.0;
short   *ptr;

best_pitch = min_pitch;
ptr = pbuf + PBUF_SIZE - min_pitch;
syy = 1.0;
for (i = 0; i < len; i++)
{
        syy += = (*ptr) * (*ptr);
        ptr++;
}
for (j = min_pitch; j < max_pitch; j++)
{
    sxy = 0.0;
    ptr = pbuf + PBUF_SIZE - j;
    for (i = 0; i < len; i++)
        sxy += = x[i] * (*ptr++);

    if (sxy > 0 && (sxy * sxy * best_syy > best_sxy * best_sxy * syy))
    {
        best_syy = syy;
        best_sxy = sxy;
        best_pitch = j;
    }
    syy = syy - pbuf[PBUF_SIZE - j + len - 1] * pbuf[PBUF_SIZE - j + len - 1]
            + pbuf[PBUF_SIZE - j - 1] * pbuf[PBUF_SIZE - j - 1];
}

*pitch = best_pitch;
*beta = best_sxy / best_syy;
}


/* Quantization of LTP gain parameter */
CodePitchFilterGain(beta, bcode)
float beta;
unsigned int *bcode;
{
    int i;
    for (i = 0; i < DLB_TAB_SIZE; i++)
    {
        if (beta <= dlb_tab[i])
            break;
    }
    *bcode = i;
}


/* Pitch filter */
PitchFilter(data, len, pbuf, pitch, ibeta)
float   *data;
```

```
        short   ibeta;
        short   *pbuf;
        short      len;
        unsigned int pitch;
        {

            long           pn;
            int        i, j;


            j = PBUF_SIZE - pitch;
            for (i = 0; i < len; i++)
            {
                pn  = ((ibeta * pbuf[j++]) >> 4);
                data[i] -= pn;
            }
        }


/* Forward Noise Shaping filter */
FNSFilter(float *inp, float *state, short len, float *out)
{
        short i, j;
        for (j = 0; j < len; j++)
        {
            float tmp = inp[j];
            for (i = 1; i <= NSF_ORDER; i++)
                tmp += state[i] * nsf[i];
            out[j] = state[0] = tmp;
            for (i = NSF_ORDER; i > 0; i--)
                state[i] = state[i-1];
        }
}


/* Update Noise shaping Filter states */
UpdateNSFState(float *inp, float *state, short len)
{
        short i, j;
        float     temp_state[NSF_ORDER+1];

        for (i = 0; i <= NSF_ORDER; i++)
            temp_state[i] = 0;

        for (j = 0; j < len; j++)
        {
            float tmp = inp[j];
            for (i = 1; i <= NSF_ORDER; i++)
                tmp += temp_state[i] * nsf[i];
            temp_state[0] = tmp;
            for (i = NSF_ORDER; i > 0; i--)
                temp_state[i] = temp_state[i-1];
```

```
        }
      for (i = 0; i < = NSF_ORDER; i + +)
          state[i] = state[i] - temp_state[i];
    }


    /* Quantization of Segment Power */
    CodeBlockGain(power, gcode)
    float power;
    unsigned int *gcode;
    {
      int i;
      for (i = 0; i < DLG_TAB_SIZE; i + +)
      {
        if (power < = dlg_tab[i])
          break;
      }
      *gcode = i;
    }


    /* Full search Coder */
    VQCoder(float *x, float *nsf_state, short len, struct frame *bs)
    {
      float         max_x, tmp;
      int           i, j, k, index, lshift_count;
      unsigned int  gcode;
      float         min_err = 0;


      max_x = x[0];
      for (i = 1; i < len; i + +)
          if ( fabs(x[i]) > max_x)
              max_x = fabs(x[i]);


      CodeBlockGain(max_x, &gcode);
      max_x = qlg_tab[gcode];
      lshift_count = 7 - gcode;          /* To scale 14-bit Code book output to the 16-bit
    actual value */
      bs->gcode = gcode;


      for (i = 0; i < len; i + = vsize)
      {
          /* Filter the data vector */
          FNSFilter(&x[i], nsf_state, vsize, &x[i]);


          /* Scale data */
          for (j = i; j < i + vsize; j + +)
              x[j] = x[j] * 1024 / max_x;


          index = 0;
          for (j = 0; j < cbook_size; j + +)
          {
```

```
            tmp = EncodeBook[j][vsize] * 1024.0;
            for (k = 0; k < vsize; k++)
                tmp -= x[i+k] * EncodeBook[j][k];

            if (tmp < min_err || j == 0)
            {
                index = j;
                min_err = tmp;
            }
        }
        bs->vqcode[i/vsize] = index;

        /* Rescale data: Decoded data is 14-bits, convert to 16 bits */
        if (lshift_count)
        {
            for (k = 0; k < vsize; k++)
                x[i+k] = ((4 * DecodeBook[index][k]) >> lshift_count);
        }
        else
        {
            for (k = 0; k < vsize; k++)
                x[i+k] = 4 * DecodeBook[index][k];
        }

        /* Update noise shaping filter state */
        UpdateNSFState(&x[i], nsf_state, vsize);
    }
}

init_compress()
{
    int i;
    oc_state[0] = 0;;
    oc_state[1] = 0;;
    for (i = 0; i <= PORDER; i++)
        pstate[i] = dstate[i] = 0;
    for (i = 0; i < PBUF_SIZE; i++)
        AnaPbuf[i] = 0;
    for (i=0; i <= NSF_ORDER; i++)
        nsf_state[i] = 0;
}

Encoder(xn, frame_size, min_pitch, max_pitch, bs)
short xn[];
struct frame *bs;
short   frame_size, min_pitch, max_pitch;
{
    unsigned int pitch, bcode;
    float   preemp_xn[PBUF_SIZE], beta;
    short   xn_copy[PBUF_SIZE];
```

```
short    ibeta;
float    acc;
int i, j;
```

```
/* Offset Compensation */
for (i = 0; i < frame_size; i++)
{
    float inp = xn[i];
    xn[i] = inp - oc_state[0] + ALPHA * oc_state[1];
    oc_state[1] = xn[i];
    oc_state[0] = inp;
}
```

```
/* Linear Prediction Filtering */
for (i = 0; i < frame_size; i++)
{
    acc = pstate[0] = xn[i];
    for (j = 1; j <= PORDER; j++)
        acc -= pstate[j] * pfilt[j];
    xn_copy[i] = preemp_xn[i] = acc;
    for (j = PORDER; j > 0; j--)
        pstate[j] = pstate[j-1];
}
```

```
GetPitchFilterPars (xn_copy, frame_size, AnaPbuf, min_pitch,
    max_pitch, &pitch, &beta);
CodePitchFilterGain(beta, &bcode);
ibeta = qlb_tab[bcode];
```

```
bs->bcode = bcode;
bs->pitch = pitch - min_pitch + 1;
```

```
PitchFilter(preemp_xn, frame_size, AnaPbuf, pitch, ibeta);
```

```
VQCoder(preemp_xn, nsf_state, frame_size, bs);
```

```
/* Inverse Filtering */
j = PBUF_SIZE - pitch;
for (i = 0; i < frame_size; i++)
{
    xn_copy[i] = preemp_xn[i];
    xn_copy[i] += ((ibeta * AnaPbuf[j++]) >> 4);
}
```

```
/* Update Pitch Buffer */
j = 0;
for (i = frame_size; i < PBUF_SIZE; i++)
    AnaPbuf[j++] = AnaPbuf[i];
for (i = 0; i < frame_size; i++)
```

```
        AnaPbuf[j++] = xn_copy[i];

    /* Inverse LP filtering */
    for (i = 0; i < frame_size; i++)
    {
        acc = xn_copy[i];
        for (j = 1; j <= PORDER; j++)
            acc = acc + dstate[j] * pfilt[j];
        dstate[0] = acc;
        for (j = PORDER; j > 0; j--)
            dstate[j] = dstate[j-1];
    }

    for (j = 0; j <= PORDER; j++)
        pstate[j] = dstate[j];
}


compress (short *input, short ilen, unsigned char *output, long *olen, long docomp)
{
    int             i, j, vcount;
    unsigned char   temp;
    short           frame_size, min_pitch, max_pitch;

    if (docomp > 2)
    {
        init_compress();

        if (NOMINAL_PITCH < 165)
        {
            min_pitch = 96;
            frame_size = 96;
            max_pitch = 350;
        }
        else
        {
            min_pitch = 160;
            frame_size = 160;
            max_pitch = 414;
        }

        bs_size = frame_size / vsize + 2;
        /* TEMPORARY: Storing State information */
        pstate[1] = *(input - 1);
        if (pstate[1] > 0)
            pstate[1] = (pstate[1] + 128) / 256 + 128;
        else
            pstate[1] = (pstate[1] - 128) / 256 + 128;

        if (pstate[1] < 0)
            pstate[1] = 0;
```

```
          if (pstate[1] > 255)
              pstate[1] = 255;
          *output = pstate[1];
          j = 1;
          pstate[1] = pstate[1] - 128;
          pstate[1] = 256 * pstate[1];
          dstate[1] = pstate[1];
          /* End of Hack */
          for (i = 0; i < ilen; i + = frame_size)
          {
              Encoder(input + i, frame_size, min_pitch, max_pitch, output + j);
              j + = bs_size;
          }
          j - = bs_size;

          /* Number of vectors in last frame */
          vcount = (ilen + frame_size - i + vsize - 1) / vsize;
          temp = output[j];
          output[j] = vcount + LAST_FRAME_FLAG;
          output[j + vcount + 2] = temp;
          *olen = j + vcount + 3;
      }
      else
      {
          static long SampCount = 0;
          copy(input, output, 2*ilen);
          SampCount + = ilen;
          *olen = ilen;
      }
  }

copy(a, b, len)
short  *a, *b;
short  len;
{
    int i;
    for (i = 0; i < len; i + +)
        *b+ + = (*a+ +);
}
```

## II. DECODER MODULE

[0135]

```
#include <Types.h>
#include <Memory.h>
#include <Quickdraw.h>
#include <ToolUtils.h>
#include <errors.h>
#include <files.h>

#include "vtcint.h"
#include <stdlib.h>
#include <math.h>
#include <sysequ.h>
#include <string.h>

#define MAX_CBOOK_SIZE          256
#define     LAST_FRAME_FLAG     128
#define     PORDER                1
#define     IPCONS                7               /* 7/8 */

#define     LARGE_NUM                 100000000
#define     VOICED    1

#define LEFT                  0
#define     RIGHT             1
#define     UNVOICED          0

#define     PFILT_ORDER             8

struct frame {
  unsigned  gcode : 4;
  unsigned       bcode : 4;
  unsigned pitch : 8;
  unsigned char vqcode[];
};

void expand(short **DecodeBook, short frame_size, short vsize,
    short min_pitch, struct frame *bs, short *output, short smpnum);

get_compr_pars(short ratio, short frame_size, short *vsize,
    short *cbook_size, short *bs_size)
{
  switch (ratio)
  {
      case 4:
          *vsize = 2;
          *cbook_size = 256;
          *bs_size = frame_size/2 + 2;
          break;
```

```
            case 7:
                *vsize = 4;
                *cbook_size = 256;
                *bs_size = frame_size/4 + 2;
                break;
            case 14:
                *vsize = 8;
                *cbook_size = 256;
                *bs_size = frame_size/8 + 2;
                break;
            case 24:
                *vsize = 16;
                *cbook_size = 256;
                *bs_size = frame_size/16 + 2;
                break;
            default:
                *vsize = 2;
                *cbook_size = 256;
                *bs_size = frame_size/2 + 2;
                break;
        }
    }

short *SnInit(short comp_ratio)
{
    short *state, *ptr;
    int i;

    state = ptr = (short*)NewPtr((PFILT_ORDER + 1  + PFILT_ORDER/2 + 2) *
sizeof(short));
    if ( state = = nil )
    {
        return nil;
    }
    for (i = 0;i < PFILT_ORDER + 1;i + +)
            *ptr + + = 0;
/*
    if (comp_ratio = = 24)
    {
        *ptr + + = 0.036953 * 32768 + 0.5;
        *ptr + + = -0.132232 * 32768 - 0.5;
        *ptr + + = 0.047798 * 32768 + 0.5;
        *ptr + + = 0.403220 * 32768 + 0.5;
        *ptr + + = 0.290033 * 32768 + 0.5;
    }
    else
    {
        *ptr + + = 0.074539 * 32768 + 0.5;
        *ptr + + = -0.174290 * 32768 - 0.5;
        *ptr + + = 0.013704 * 32768 + 0.5;
```

```
                        *ptr + +  = 0.426815 * 32768 + 0.5;
                        *ptr + +  = 0.320707 * 32768 + 0.5;
                   }
             */
             if (comp_ratio = = 24)
             {
                   *ptr + +  = 1211;
                   *ptr + +  = -4333;
                   *ptr + +  = 1566;
                   *ptr + +  = 13213;
                   *ptr + +  = 9504;
             }
             else
             {
                   *ptr + +  = 2442;
                   *ptr + +  = -5711;
                   *ptr + +  = 449;
                   *ptr + +  = 13986;
                   *ptr + +  = 10509;
             }
             *ptr = 0;          /* DC value */
             return state;
        }


        SnDone(char *state)
        {
           if ( state ! = nil )
           {
                DisposPtr(state);
           }
        }


        short **SnDeInit(p, ratio, frame_size)
        short *p,ratio, frame_size;
        {
           int i;
           short cbook_size = 256, vsize = 16, bs_size;
           short **DecodeBook;

           get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);

           DecodeBook = (short**)NewPtr(cbook_size * sizeof(short*));
           if (DecodeBook) {
                for (i = 0; i < cbook_size; i + +)
                {
                     DecodeBook[i] = p;
                     p + = vsize;
                }
           }
           return DecodeBook;
```

```
}

SnDeDone(char *DecodeBook)
{
    if ( DecodeBook != nil )
    {
        DisposPtr(DecodeBook);
    }
}


void
expand(short **DecodeBook, short frame_size, short vsize,
    short min_pitch, struct frame *bs, short *output, short smpnum)
{
    short    count;
    short    *bptr, *sptr1, *sptr2;
    unsigned short pitch, bcode;
/*
    short qlb_tab[] = {
    1, 2, 3, 4, 5, 6, 7, 8,
    9, 10, 11, 12, 13, 14, 15, 16
    };
*/
    bcode = bs->bcode;
    pitch = bs->pitch + min_pitch - 1;

    /* Decode VQ vectors */
    {
        unsigned    char    *cptr;
        short    k, vsize_by_2;
        short    rshift_count = 7 - bs->gcode;   /* We want the output to be 14-bit
number */

        sptr1 = output + smpnum;
        cptr = bs->vqcode;
        vsize_by_2 = (vsize >> 1) + 1; /* +1 since we do a while (--i) instead of
while (i--) */
        if (rshift_count)
        {
            for (k = 0; k < frame_size; k += vsize)
            {
                bptr = DecodeBook[*cptr++];
                count = vsize_by_2;
                while (--count)
                {
                    *sptr1++ = ((*bptr++) >> rshift_count);
                    *sptr1++ = ((*bptr++) >> rshift_count);
                }
            }
        }
```

```
        else
        {
            for (k = 0; k < frame_size; k + = vsiz )
            {
                bptr = DecodeBook[*cptr+ +];
                count = vsize_by_2;
                while (--count)
                {
                    *sptr1 + +  =  *bptr+ +;
                    *sptr1 + +  =  *bptr+ +;
                }
            }
        }
    }

    /* Inverse Filtering */
    if (smpnum < pitch)
    {
        sptr1 = output + pitch;
        count = smpnum + frame_size + 1 - pitch; /* +1 since we do a while (--i)
instead of while (i--) */
        sptr2 = sptr1 - pitch;
        switch (bcode)
        {
            case 0:
                while (--count)
                    *sptr1 + + + = ((*sptr2+ +) > > 4);
                break;
            case 1:
                while (--count)
                    *sptr1 + + + = ((*sptr2+ +) > > 3);
                break;
            case 2:
                while (--count)
                    *sptr1 + + + = ((3 * (*sptr2+ +)) > > 4);
                break;
            case 3:
                while (--count)
                    *sptr1 + + + = ((*sptr2+ +) > > 2);
                break;
            case 4:
                while (--count)
                    *sptr1 + + + = ((5 * (*sptr2+ +)) > > 4);
                break;
            case 5:
                while (--count)
                    *sptr1 + + + = ((3 * (*sptr2+ +)) > > 3);
                break;
            case 6:
                while (--count)
```

```
                        *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 4);
                    break;
                case 7:
                    while (--count)
                        *sptr1 + +  + = ((*sptr2 + +) > > 1);
                    break;
                case 8:
                    while (--count)
                    {
                        long    tmp;
                        tmp = *sptr2 + +;
                        *sptr1 + +  + = (((tmp < < 3) + tmp) > > 4);
                    }
                    break;
                case 9:
                    while (--count)
                        *sptr1 + +  + = ((5 * (*sptr2 + +)) > > 3);
                    break;
                case 10:
                    while (--count)
                    {
                        long    tmp;
                        tmp = *sptr2 + +;
                        *sptr1 + +  + = (((tmp < < 3) + 3 * tmp) > > 4);
                    }
                    break;
                case 11:
                    while (--count)
                        *sptr1 + +  + = ((3 * (*sptr2 + +)) > > 2);
                    break;
                case 12:
                    while (--count)
                    {
                        long    tmp;
                        tmp = *sptr2 + +;
                        *sptr1 + +  + = (((tmp < < 4) - 3 * tmp) > > 4);
                    }
                    break;
                case 13:
                    while (--count)
                        *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 3);
                    break;
                case 14:
                    while (--count)
                    {
                        long    tmp;
                        tmp = *sptr2 + +;
                        *sptr1 + +  + = (((tmp < < 4) - tmp) > > 4);
                    }
                    break;
```

```
                case 15:
                    while (--count)
                        *sptr1++ += = *sptr2++;
                    break;
            }
        } else {
            sptr1  = output + smpnum;
            sptr2  = sptr1 - pitch;
            count  = (frame_size / 4) + 1;
            switch (bcode)
            {
                case 0:
                    while (--count) {
                        *sptr1++ += = ((*sptr2++) >> 4);
                        *sptr1++ += = ((*sptr2++) >> 4);
                        *sptr1++ += = ((*sptr2++) >> 4);
                        *sptr1++ += = ((*sptr2++) >> 4);
                    }
                    break;
                case 1:
                    while (--count) {
                        *sptr1++ += = ((*sptr2++) >> 3);
                        *sptr1++ += = ((*sptr2++) >> 3);
                        *sptr1++ += = ((*sptr2++) >> 3);
                        *sptr1++ += = ((*sptr2++) >> 3);
                    }
                    break;
                case 2:
                    while (--count) {
                        *sptr1++ += = ((3 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((3 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((3 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((3 * (*sptr2++)) >> 4);
                    }
                    break;
                case 3:
                    while (--count) {
                        *sptr1++ += = ((*sptr2++) >> 2);
                        *sptr1++ += = ((*sptr2++) >> 2);
                        *sptr1++ += = ((*sptr2++) >> 2);
                        *sptr1++ += = ((*sptr2++) >> 2);
                    }
                    break;
                case 4:
                    while (--count) {
                        *sptr1++ += = ((5 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((5 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((5 * (*sptr2++)) >> 4);
                        *sptr1++ += = ((5 * (*sptr2++)) >> 4);
                    }
```

```
            break;
        case 5:
            while (--count) {
                *sptr1++ += ((3 * (*sptr2++)) >> 3);
                *sptr1++ += ((3 * (*sptr2++)) >> 3);
                *sptr1++ += ((3 * (*sptr2++)) >> 3);
                *sptr1++ += ((3 * (*sptr2++)) >> 3);
            }
            break;
        case 6:
            while (--count) {
                *sptr1++ += ((7 * (*sptr2++)) >> 4);
                *sptr1++ += ((7 * (*sptr2++)) >> 4);
                *sptr1++ += ((7 * (*sptr2++)) >> 4);
                *sptr1++ += ((7 * (*sptr2++)) >> 4);
            }
            break;
        case 7:
            while (--count) {
                *sptr1++ += ((*sptr2++) >> 1);
                *sptr1++ += ((*sptr2++) >> 1);
                *sptr1++ += ((*sptr2++) >> 1);
                *sptr1++ += ((*sptr2++) >> 1);
            }
            break;
        case 8:
            while (--count) {
                long    tmp;
                tmp = *sptr2++;
                *sptr1++ += ((8 * tmp + tmp) >> 4);
                tmp = *sptr2++;
                *sptr1++ += ((8 * tmp + tmp) >> 4);
                tmp = *sptr2++;
                *sptr1++ += ((8 * tmp + tmp) >> 4);
                tmp = *sptr2++;
                *sptr1++ += ((8 * tmp + tmp) >> 4);
            }
            break;
        case 9:
            while (--count) {
                *sptr1++ += ((5 * (*sptr2++)) >> 3);
                *sptr1++ += ((5 * (*sptr2++)) >> 3);
                *sptr1++ += ((5 * (*sptr2++)) >> 3);
                *sptr1++ += ((5 * (*sptr2++)) >> 3);
            }
            break;
        case 10:
            while (--count) {
                long tmp;
                tmp = *sptr2++;
```

```
                    *sptr1 + +  + = (((tmp < < 3) + 3 * tmp) > > 4);
                    tmp = *sptr2 + +;
                    *sptr1 + +  + = (((tmp < < 3) + 3 * tmp) > > 4);
                    tmp = *sptr2 + +;
                    *sptr1 + +  + = (((tmp < < 3) + 3 * tmp) > > 4);
                    tmp = *sptr2 + +;
                    *sptr1 + +  + = (((tmp < < 3) + 3 * tmp) > > 4);
                }
            break;
        case 11:
            while (--count) {
                *sptr1 + +  + = ((3 * (*sptr2 + +)) > > 2);
                *sptr1 + +  + = ((3 * (*sptr2 + +)) > > 2);
                *sptr1 + +  + = ((3 * (*sptr2 + +)) > > 2);
                *sptr1 + +  + = ((3 * (*sptr2 + +)) > > 2);
            }
            break;
        case 12:
            while (--count) {
                long tmp;
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - 3 * tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - 3 * tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - 3 * tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - 3 * tmp) > > 4);
            }
            break;
        case 13:
            while (--count) {
                *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 3);
                *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 3);
                *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 3);
                *sptr1 + +  + = ((7 * (*sptr2 + +)) > > 3);
            }
            break;
        case 14:
            while (--count) {
                long tmp;
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - tmp) > > 4);
                tmp = *sptr2 + +;
                *sptr1 + +  + = (((tmp < < 4) - tmp) > > 4);
            }
```

```
                    break;
                case 15:
                    while (--count) {
                        *sptr1++ += *sptr2++;
                        *sptr1++ += *sptr2++;
                        *sptr1++ += *sptr2++;
                        *sptr1++ += *sptr2++;
                    }
                    break;
            }
        }

    }

    short SnDecompress(DecodeBook, ratio, frame_size, min_pitch, bstream, output)
    short **DecodeBook, ratio;
    unsigned char *bstream;
    short *output, frame_size, min_pitch;
    {
        short    count, SampCount;
        register short    dstate;
        short    vcount;
        short    vsize, cbook_size, bs_size;

        get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);

        dstate = *bstream++;
        dstate = (dstate - 128) << 6;

        SampCount = 0;
        while((*bstream & LAST_FRAME_FLAG) == 0)
        {
            expand(DecodeBook, frame_size, vsize, min_pitch,
                (struct frame *)bstream, output, SampCount);
            bstream += bs_size;
            SampCount += frame_size;
        }
        vcount = *bstream - LAST_FRAME_FLAG;
        *bstream = *(bstream + 2 + vcount);
        expand(DecodeBook, frame_size, vsize, min_pitch,
            (struct frame *)bstream, output, SampCount);
        *bstream = vcount + LAST_FRAME_FLAG;
        SampCount += vcount * vsize;

        count = (SampCount >> 1) + 1;
        while (--count) {
            *output++ = dstate = ((IPCONS * dstate) >> 3) + *output;
            *output++ = dstate = ((IPCONS * dstate) >> 3) + *output;
        }
        output -= SampCount;
```

```
        return SampCount;
    }

#define    FILTER  state + PFILT_ORDER + 1
#define DC_VAL    state + PFILT_ORDER + PFILT_ORDER/2 + 2
void SnSampExpandFilt(short *src, short off, short len,
    char *dest,short *state)
{
    short        input, temp;
    long         acc;
    register short dc = *(DC_VAL);
    register short *sptr1, *sptr2;

    src += off;
    len++;
    sptr1 = state;
    sptr2 = state + PFILT_ORDER;
    while (--len) {
        input = *src++ - dc;
        dc += input >> 5;

        temp = input + *sptr1++; /* (state[0] + state[8]) * filter[0] */
        acc = temp * *(FILTER);

        temp = *--sptr2 + *sptr1++;  /* (state[1] + state[7]) * filter[1] */
        acc += temp * *(FILTER+1);

        temp = *--sptr2 + *sptr1++;  /* (state[2] + state[6]) * filter[2] */
        acc += temp * *(FILTER+2);

        temp = *--sptr2 + *sptr1++;  /* (state[3] + state[5]) * filter[3] */
        acc += temp * *(FILTER+3);

        acc += *sptr1 * *(FILTER+4); /* state[4] * filter[4] */

        if (acc > 0)
        {
            temp = (acc + (257 << 20)) >> 21;
            if (temp > 255)
                temp = 255;
        }
        else
        {
            temp = (acc + (255 << 20)) >> 21;
            if (temp < 0)
                temp = 0;
        }
        *dest++ = temp;
```

```
        sptr1 -= 4;
        sptr2 -= 4;
        *sptr1++ = *sptr2++;     /* state[0] = state[1] */
        *sptr1++ = *sptr2++;     /* state[1] = state[2] */
        *sptr1++ = *sptr2++;     /* state[2] = state[3] */
        *sptr1++ = *sptr2++;     /* state[3] = state[4] */
        *sptr1++ = *sptr2++;     /* state[4] = state[5] */
        *sptr1++ = *sptr2++;     /* state[5] = state[6] */
        *sptr1++ = *sptr2++;     /* state[6] = state[7] */
        *sptr1 = input;          /* state[7] = input */
        sptr1 -= 7;
    }
    *(DC_VAL) = dc;
}
```

### III. BLENDING MODULE

[0136]

```
/* A module for blending two diphones */

typedef struct {
    short lptr, pitch;
    short weight, weight_inc;
} bstate;

void SnBlend(pitchp lp, pitchp rp, short cur_tot, short tot,
    short type, bstate *bs)
{
#pragma unused (tot)

    short    count;
    short    *ptr1, *ptr2;

    if (type = = VOICED)
    {
        if (cur_tot)
            return;
        {
            short    weight;
            long     min_amdf;
            short    best_lag = 0, lag;
            short    window_size;
            short    weight_inc;

            /* First replicate the left pitch period */
            ptr1 = lp->bufp;
            ptr2 = ptr1 + lp->olen;
            count = lp->olen + 1;
            while (--count)
                *ptr2++ = *ptr1++;

            /* Smooth the discontinuity */
            {
                register short en, e2;

                en = lp->bufp[2] +
                    3 * (lp->bufp[0] - lp->bufp[1]) - lp->bufp[lp->olen - 1];

                e2 = lp->bufp[0] - lp->bufp[lp->olen - 1];

                if (en * en > e2 * e2)
                    en = e2;
```

```
    ptr2  =  lp->bufp  +  lp->olen;
    count  =  (lp->olen  > >  1)  +  1:
    while (--count)
    {
        *--ptr2  + = en;
        en  =  (((en  < <  4) - en)  > >  4);
    }
}

min_amdf  =  LARGE_NUM;

window_size  =  rp->olen;
if (lp->olen  <  rp->olen)
    window_size  =  lp->olen;

lag  =  rp->olen;
while (--lag)
{
    long amdf  =  0;
    ptr1  =  rp->bufp;
    ptr2  =  lp->bufp  +  lag;
    count  =  ((window_size + 3)  > >  2)  +  1;
    while (--count)
    {
        short tmp;
        tmp  =  (*ptr1 - *ptr2);
        if (tmp  >  0)
            amdf  + = tmp;
        else
            amdf - = tmp;
        ptr1  + = 4;
        ptr2  + = 4;
    }
    if (amdf  <  min_amdf)
    {
        best_lag  =  lag;
        min_amdf  =  amdf;
    }
}

bs->pitch  =  lp->olen;
/* Update left buffer */
if (best_lag  <  (lp->olen  > >  1))
{
    /* Add best_lag samples to the length of left pulse*/
    lp->olen  + = best_lag;
}
else
{
    /* Delete a few samples from the left pulse */
```

```
                    lp->olen = best_lag;
                }
            bs->lptr = best_lag;
            weight_inc = 32767/window_size;
            weight = 32767 - weight_inc;

            ptr1 = rp->bufp;
            ptr2 = lp->bufp + bs->lptr;
            count = window_size + 1;
            while (--count)
            {
                *ptr1++ += (((short) (*ptr2++ - *ptr1) * weight) >> 15);
                weight -= weight_inc;
            }
        }
    }
    else
    {
        register short    delta;

        /* Just blend 15 samples */
        ptr2 = lp->bufp + lp->olen - 15;
        ptr1 = rp->bufp;
/*
        for (i = 1; i < 16; i++)
        {
            *ptr1 = *ptr2 + (i * (*ptr1 - *ptr2)) >> 4;
            ptr1++;
            ptr2++;
        }
*/
        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (delta >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((delta) >> 3);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((3 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (delta >> 2);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((5 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((3 * delta) >> 8);

        delta = *ptr1 - *ptr2;
```

```
*ptr1++ = *ptr2++ + ((7 * delta) >> 4);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + (delta >> 1);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + (((delta << 3) + delta) >> 4);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + ((5 * delta) >> 3);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + (((delta << 3) + 3 * delta) >> 4);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + ((3 * delta) >> 2);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + (((delta << 4) - 3 * delta) >> 4);

delta = *ptr1 - *ptr2;
*ptr1++ = *ptr2++ + ((7 * delta) >> 3);

delta = *ptr1 - *ptr2;
*ptr1 = *ptr2 + (((delta << 4) - delta) >> 4);


    lp->olen -= 15;
    }
}
```

## IV. INTONATION ADJUSTMENT MODULE

[0137]

```
/* A module for deleting a pitch period */
/*
    Pointer src1 points to Left Pitch period
    Pointer src2 points to Right Pitch period
    Pointer dst points to Resulting Pitch period
    len = length of the pitch periods
*/
skip_pulses(short *src1, short *src2, short *dst, short len)
{
    short i;
    register short    weight, cweight;

    i = len + 1;
    weight = cweight = 32767/i;
    while (--i)
    {
        *dst++ = *src1++ + (((short) (*src2++ - *src1) * cweight) >> 15);
        cweight += weight;
    }
}


/* A module for Inserting a pitch period */
/*
    Locn buffer[curbeg] points to Left Pitch period
    Locn buffer[curbeg+curlen] points to Right Pitch period
    Pointer dst points to Resulting Pitch period
    curlen = length of the pitch periods
*/
insert_pulse(short *buffer, short *dst, short curlen,  short curbeg)
{
    short    weight, cweight, count;
    short    *src1, *src2;

    src1 = buffer + curbeg;
    src2 = buffer + curbeg + curlen;
    weight = 32767 / curlen;
    cweight = weight;
    count = curlen + 1;
    while (--count)
    {
        *dst++ = *src1++ = *src2++ + (((short) (*src1 - *src2) * cweight) >>
15);
        cweight += weight;
    }
}

    /* This module is used to change pitch information in the concatenated speech */
```

```
// This routine depends on the desired length (deslen) being at least half
// and no more than twice the actual length (len).

    void SnChangePitch(short *buf, short *next, short len, short deslen,short lvoc,short
rvoc,short dosmooth)
    {
#pragma unused(rvoc, dosmooth)
        short    delta;
        short    count;
        short    *bptr, *aptr;
        short    weight, weight_inc;
        if (!lvoc || (deslen = = len)) return;

        if (deslen > len)
        {
            /* Increase Pitch period */
            delta = deslen - len;
            bptr = buf + len;
            aptr = buf + deslen;
            count = delta + 1;
            while (--count)
                *--aptr = *--bptr;

            count = len - delta + 1;
            weight = weight_inc = 32767 / count;
            while (--count)
            {
                register short tmp2;
                tmp2 = (*--aptr - *--bptr);
                *aptr = *bptr + ((tmp2 * weight) > > 15);
                weight + = weight_inc;
            }
            return;
        }
        {
            /* Shorten Pitch Period */
            short wsize;

            delta = len - deslen;
            wsize = 2 * delta;

            if (wsize > deslen)
                wsize = deslen;

            weight_inc = 32767 / (wsize + 1);
            weight = weight_inc;
            aptr = buf + deslen;
            bptr = buf + len - wsize;
            count = wsize - delta + 1:
```

```
while (--count)
{
    *bptr++ += (((short) (*aptr++ - *bptr) * weight ) >> 15);
    weight += weight_inc;
}
aptr = buf + deslen;
bptr = next;
count = delta + 1;
weight = 32767 - weight;
while (--count)
{
    *bptr++ += (((short) (*aptr++ - *bptr) * weight ) >> 15);
    weight -= weight_inc;
}
}
}
```

## Claims

1. An apparatus for concatenating a first digital frame of N samples having respective magnitudes representing a first quasi-periodic waveform and a second digital frame of M samples having respective magnitudes representing a second quasi-periodic waveform, comprising:

   a buffer (15) to store the samples of first and second digital frames;
   means, coupled to the buffer store, for determining a blend point for the first and second digital frames in response to magnitudes of samples in the first and second digital frames;
   blending means, coupled with the buffer store and the means for determining, for computing a digital sequence representing a concatenation of the first and second quasi-periodic waveforms in response to the first frame, the second frame and the blend point.

2. The apparatus of claim 1, further including:
   transducer means, coupled to the blending means, for transducing the digital sequence to an analog concatenated waveform.

3. The apparatus of claims 1 or 2, wherein the means for determining includes:

   first means for computing an extended frame in response to the first digital frame;
   second means for finding a subset of the extended frame which matches the second digital frame relatively well, and defining the blend point as a sample in the subset.

4. The apparatus of claim 3, wherein the extended frame comprises a concatenation of the first digital frame with a replica of the first digital frame.

5. The apparatus of claim 3 or 4, wherein the subset of the extended frame which matches the second digital frame relatively well is a subset with a minimum average magnitude difference over the samples in the subset, and the blend point is a first sample in the subset.

6. The apparatus of any preceding claim, wherein the means for determining includes:

   first means for computing an extended frame comprising a discontinuity-smoothed concatenation of the first digital frame with a replica of the first digital frame;
   second means for finding a subset of the extended frame with a minimum average magnitude difference between the samples in the subset and the second digital frame, and defining the blend point as a first sample

in the subset.

7. The apparatus of any preceding claim, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and

means for combining the second digital frame with a second set of samples derived from the first digital frame and the blend point, with emphasis on the second set in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

8. The apparatus of claim 6, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and

means for combining the second digital frame with the subset of the extended frame, with emphasis on the subset of the extended frame in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

9. The apparatus of claim 8, wherein the first and second digital frames represent endings and beginnings respectively of adjacent diphones in speech, and further including:

transducer means, coupled to the blending means, for transducing the digital sequence to a sound in speech synthesis.

10. An apparatus for concatenating a first digital frame of N samples having respective magnitudes representing a first sound segment and a second digital frame of M samples having respective magnitudes representing a second sound segment, comprising:

a buffer store to store the samples of first and second digital frames;

means, coupled to the buffer store, for determining a blend point for the first and second digital frames in response to magnitudes of samples in the first and second digital frames;

blending means, coupled with the buffer store and the means for determining, for computing a digital sequence representing a concatenation of the first and second sound segments in response to the first frame, the second frame and the blend point; and

transducer means, coupled to the blending means, for transducing the digital sequence to sound.

11. The apparatus of claim 10, wherein the means for determining includes:

first means for computing an extended frame in response to the first digital frame;

second means for finding a subset of the extended frame which matches the second digital frame relatively well, and defining the blend point as a sample in the subset.

12. The apparatus of claim 11, wherein the extended frame comprises a concatenation of the first digital frame with a replica of the first digital frame.

13. The apparatus of claim 11 or 12, wherein the subset of the extended frame which matches the second digital frame relatively well is a subset with a minimum average magnitude difference over the samples in the subset, and the blend point is a first sample in the subset.

14. The apparatus of any of claims 10 to 13, wherein the means for determining includes:

first means for computing an extended frame comprising a discontinuity-smoothed concatenation of the first digital frame with a replica of the first digital frame;

second means for finding a subset of the extended frame with a minimum average magnitude difference between the samples in the subset and the second digital frame, and defining the blend point as a first sample in the subset.

15. The apparatus of any of claims 10 to 14, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and

means for combining the second digital frame with a second set of samples derived from the first digital frame and the blend point, with emphasis on the second set in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

16. The apparatus of claim 14, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and

means for combining the second digital frame with the subset of the extended frame, with emphasis on the subset of the extended frame in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

17. The apparatus of claim 16, wherein the first and second digital frames represent endings and beginnings respectively of adjacent diphones in speech, and the transducer means produces synthesized speech.

18. An apparatus for synthesizing speech in response to a text, comprising:

means (21) for translating text to a sequence of sound segment codes;

means (23), responsive to sound segment codes in the sequence, for decoding the sequence of sound segment codes to produce strings of digital frames of a plurality of samples representing sounds for respective sound segment codes in the sequence, wherein the identified strings of digital frames have beginnings and endings;

means (24) for concatenating a first digital frame at the ending of an identified string of digital frames of a particular sound segment code in the sequences with a second digital frame at the beginning an identified string of digital frames of an adjacent sound segment code in the sequence to produce a speech data sequence, including

a buffer store to store the samples of first and second digital frames;

means, coupled to the buffer store, for determining a blend point for the first and second digital frames in response to magnitudes of samples in the first and second digital frames; and

blending means, coupled with the buffer store and the means for determining, for computing a digital sequence representing a concatenation of the first and second sound segments in response to the first frame, the second frame and the blend point; and

an audio transducer (27), coupled to the means for concatenating, to generate synthesized speech in response to the speech data sequence.

19. The apparatus of claim 18, further including:

means (25,26), responsive to the sound segment codes for adjusting pitch and duration of the identified strings of digital frames in the speech data sequence.

20. The apparatus of claim 18 or 19, wherein the means for determining includes:

first means for computing an extended frame in response to the first digital frame;

second means for finding a subset of the extended frame which matches the second digital frame relatively well, and defining the blend point as a sample in the subset.

21. The apparatus of claim 20, wherein the extended frame comprises a concatenation of the first frame with a replica of the first digital frame.

22. The apparatus of claim 20 or 21, wherein the subset of the extended frame which matches the second digital frame relatively well comprises a subset with a minimum average magnitude difference over the samples in the subset, and the blend point comprises a first sample in the subset.

23. The apparatus of any of claims 18 to 22, wherein the means for determining includes:

first means for computing an extended frame comprising a discontinuity-smoothed concatenation of the first digital frame with a replica of the first digital frame;

second means for finding a subset of the extended frame with a minimum average magnitude difference between the samples in the subset and the second digital frame, and defining the blend point as a first sample in the subset.

24. The apparatus of any of claims 18 to 23, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and
means for combining the second digital frame with a second set of samples derived from the first digital frame and the blend point, with emphasis on the second set in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

25. The apparatus of claim 23, wherein the blending means includes:

means for supplying a first set of samples derived from the first digital frame and the blend point as a first segment of the digital sequence; and
means for combining the second digital frame with the subset of the extended frame, with emphasis on the subset of the extended frame in a starting sample and emphasis on the second digital frame in an ending sample to produce a second segment of the digital sequence.

26. The apparatus of any of claims 18 to 25, wherein the sound segment codes represent speech diphones, and the first and second digital frames represent endings and beginnings respectively of adjacent diphones in speech.


**Patentansprüche**

1. Vorrichtung zur Verkettung eines ersten digitalen Rahmens von N Proben mit jeweiligen Beträgen, welche eine erste quasiperiodische Wellenform darstellen, und eines zweiten digitalen Rahmens von M Proben mit jeweiligen Beträgen, welche eine zweite quasiperiodische Wellenform darstellen, mit:

   - einem Puffer (15) zum Speichern der Proben des ersten und zweiten digitalen Rahmens;
   - Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Mischungspunktes für den ersten und den zweiten digitalen Rahmen, ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;
   - Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und der zweiten quasiperiodischen Wellenform ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt.

2. Vorrichtung nach Anspruch 1, ferner mit:

   - Wandlermitteln, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in eine analoge verkettete Wellenform.

3. Vorrichtung nach einem der Ansprüche 1 oder 2, bei welcher die Mittel zur Bestimmung aufweisen:

   - erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
   - zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, und zur Definierung des Vermischungspunktes als einer Probe in der Teilmenge.

4. Vorrichtung nach Anspruch 3, bei welcher der erweiterte Rahmen eine Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

5. Vorrichtung nach einem der Ansprüche 3 oder 4, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, ein Teilmenge mit einer minimalen mittleren bzw. durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge ist, und der Vermischungspunkt eine erste Probe in der Teilmenge ist.

6. Vorrichtung nach einem der vorstehenden Ansprüche, bei welcher die Mittel zur Bestimmung aufweisen:

   - erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
   - zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge und dem zweiten digitalen Rahmen, und zur Definierung eines Vermischungspunktes als einer ersten Probe in der Teilmenge.

7. Vorrichtung nach einem der vorstehenden Ansprüche, bei welcher die Vermischungsmittel aufweisen:

   - Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben abgeleitet von dem ersten digitalen Rahmen und dem Vermischungspunkt als ein erstes Segment der digitalen Sequenz; und
   - Mittel zur Kombination des zweiten digitalen Rahmens mit einer zweiten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, unter Betonung der zweiten Menge in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

8. Vorrichtung nach Anspruch 6, bei welcher die Vermischungsmittel aufweisen:

   - Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind als ein erstes Segment der digitalen Sequenz; und
   - Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, unter Betonung der Teilmenge des erweiterten Rahmens in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

9. Vorrichtung nach Anspruch 8, bei welcher der erste und der zweite digitale Rahmen Enden bzw. Anfänge von benachbarten Diphonen bei der Sprache darstellen, und ferner aufweisen:

   - Wandlermittel, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in einen Laut bei der Sprachsynthese.

10. Vorrichtung zur Verkettung eines ersten digitalen Rahmens von N Proben mit jeweiligen Beträgen, welche ein erstes Lautsegment darstellen, und eines zweiten digitalen Rahmens von M Proben mit jeweiligen Beträgen, welche ein zweites Lautsegment darstellen, mit:

    - einem Pufferspeicher zum Speichern der Proben des ersten und des zweiten digitalen Rahmens;
    - Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Vermischungspunktes für den ersten und den zweiten digitalen Rahmen ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;
    - Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und der zweiten Lautsegmente ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt; und
    - Wandlermitteln, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in Laute.

11. Vorrichtung nach Anspruch 10, bei welcher die Mittel zur Bestimmung aufweisen:

    - erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
    - zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, und zur Definierung des Vermischungspunktes als einer Probe in der Teilmenge.

12. Vorrichtung nach Anspruch 11, bei welcher der erweiterte Rahmen eine Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

13. Vorrichtung nach einem der Ansprüche 11 oder 12, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, eine Teilmenge mit einer minimalen durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge ist, und der Vermischungspunkt eine erste Probe in der

Teilmenge ist.

**14.** Vorrichtung nach einem der Ansprüche 10 bis 13, wobei die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge und dem zweiten digitalen Rahmen und zur Definierung des Vermischungspunktes als einer ersten Probe in der Teilmenge.

**15.** Vorrichtung nach einem der Ansprüche 10 bis 14, wobei die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit einer zweiten Menge von Proben, die von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, mit Betonung der zweiten Menge in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segments der digitalen Sequenz.

**16.** Vorrichtung nach Anspruch 14, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, die von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, mit Betonung der Teilmenge des erweiterten Rahmens in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

**17.** Vorrichtung nach Anspruch 16, bei welcher der erste und der zweite digitale Rahmen Enden bzw. Anfänge benachbarter Diphone in der Sprache darstellen, und die Wandlermittel synthetisierte Sprache erzeugen.

**18.** Vorrichtung zur Synthetisierung von Sprache ansprechend auf einen Text, mit

- Mitteln (21) zur Übersetzung von Text in eine Sequenz von Lautsegmentcodierungen;
- Mitteln (23), welche ansprechend sind auf die Lautsegmentcodierungen in der Sequenz, zur Decodierung der Sequenz der Lautsegmentcodierungen zur Herstellung von Strings von digitalen Rahmen einer Anzahl von Proben, welche Laute für jeweilige Lautsegmentcodierungen in der Sequenz darstellen, wobei die identifizierten Strings der digitalen Rahmen Anfänge und Endungen bzw. Enden aufweisen;
- Mitteln (24) zur Verkettung eines ersten digitalen Rahmens an der Endung eines identifizierten Strings von digitalen Rahmen einer bestimmten Lautsegmentcodierung in den Sequenzen mit einem zweiten digitalen Rahmen am Anfang eines identifizierten Strings von digitalen Rahmen einer benachbarten Lautsequenzcodierung in der Sequenz zur Erzeugung einer Sprachdatensequenz, mit

  einem Pufferspeicher zum Speichern der Proben von ersten und zweiten digitalen Rahmen;
  Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Vermischungspunktes für den ersten und den zweiten digitalen Rahmen, ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;
  Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und zweiten Lautsegmente ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt; und
  einem Audiowandler (27), der mit den Mitteln zur Verkettung gekoppelt ist, zur Generierung synthetisierter Sprache ansprechend auf die Sprachdatensequenz.

**19.** Vorrichtung nach Anspruch 18, ferner mit:

- Mitteln, welche ansprechend auf die Lautsegmentcodierungen sind, zur Einstellung der Tonhöhe und der Dauer der identifizierten Strings der digitalen Rahmen in der Sprachdatensequenz.

**20.** Vorrichtung nach einem der Ansprüche 18 oder 19, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welcher bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist und zur Definierung des Vermischungspunktes als einer Probe in der Teilmenge.

21. Vorrichtung nach Anspruch 20, bei welcher der erweiterte Rahmen eine Verkettung des ersten Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

22. Vorrichtung nach einem der Ansprüche 20 oder 21, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des ersten digitalen Rahmens relativ gut angepaßt ist, eine Teilmenge mit einer minimalen durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge aufweist, und wobei der Vermischungspunkt eine erste Probe in der Teilmenge aufweist.

23. Vorrichtung nach einem der Ansprüche 18 bis 22, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge und dem zweiten digitalen Rahmen, und zur Definierung des Vermischungspunktes als einer ersten Probe in der Teilmenge.

24. Vorrichtung nach einem der Ansprüche 18 bis 23, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit einem zweiten Satz von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, mit Betonung der zweiten Menge in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

25. Vorrichtung nach Anspruch 23, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung eines ersten Satzes von Proben, welche abgeleitet sind von dem ersten digitalen Rahmen und dem Vermischungspunkt, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, mit Betonung der Teilmenge des erweiterten Rahmens in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

26. Vorrichtung nach einem der Ansprüche 18 bis 25, bei welcher die Lautsegmentcodierungen Sprach-Diphone darstellen, und der erste und der zweite digitale Rahmen Endungen bzw. Anfänge benachbarter Diphone in der Sprache darstellen.

## Revendications

1. Appareil pour réunir par concaténation une première trame numérique de N échantillons ayant des amplitudes respectives représentant une première forme d'onde quasi périodique et une deuxième trame numérique de M échantillons ayant des amplitudes respectives représentant une deuxième forme d'onde quasi périodique, comprenant :

une mémoire tampon (15) pour stocker les échantillons des première et deuxième trames numériques ;
des moyens, couplés à la mémoire tampon, pour déterminer un point de juxtaposition pour les première et deuxième trames numériques en réponse à des amplitudes d'échantillons dans les première et deuxième trames numériques ;
des moyens de mélange, couplés à la mémoire tampon et aux moyens de détermination, pour calculer une séquence numérique représentant une concaténation des première et deuxièmes formes d'onde quasi périodiques en réponse à la première trame, à la deuxième trame et au point de juxtaposition.

2. Appareil selon la revendication 1, comprenant en outre des moyens de transducteur, couplés aux moyens de mélange, pour convertir la séquence numérique en une forme d'onde analogique constituée par concaténation.

3. Appareil selon la revendication 1 ou 2, dans lequel les moyens de détermination comprennent :

   des premiers moyens pour calculer une trame étendue en réponse à la première trame numérique ;
   des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique, et définir le point de juxtaposition comme étant un échantillon dans le sous-ensemble.

4. Appareil selon la revendication 3, dans lequel la trame étendue comprend une concaténation de la première trame numérique avec une réplique de la première trame numérique.

5. Appareil selon la revendication 3 ou 4, dans lequel le sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique est un sous-ensemble ayant une différence moyenne minimale d'amplitude entre les échantillons dans le sous-ensemble, et le point de juxtaposition est un premier échantillon dans le sous-ensemble.

6. Appareil selon l'une quelconque des revendications précédentes, dans lequel les moyens de détermination comprennent :

   des premiers moyens pour calculer une trame étendue comprenant une concaténation à lissage de discontinuité de la première trame numérique avec une réplique de la première trame numérique ;
   des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue ayant une différence moyenne minimale d'amplitude entre les échantillons dans le sous-ensemble et la deuxième trame numérique, et définir le point de juxtaposition comme étant un premier échantillon dans le sous-ensemble.

7. Appareil selon l'une quelconque des revendications précédentes, dans lequel les moyens de mélange comprennent :

   des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et
   des moyens pour combiner la deuxième trame numérique avec un deuxième ensemble d'échantillons extrait de la première trame numérique et du point de juxtaposition, avec une mise en valeur du deuxième ensemble dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin, afin de produire un deuxième segment de la séquence numérique.

8. Appareil selon la revendication 6, dans lequel les moyens de mélange comprennent :

   des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et
   des moyens pour combiner la deuxième trame numérique avec le sous-ensemble de la trame étendue, avec une mise en valeur du sous-ensemble de la trame étendue dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin, afin de produire un deuxième segment de la séquence numérique.

9. Appareil selon la revendication 8, dans lequel les première et deuxième trames numériques représentent respectivement des commencements et des fins de diphones adjacents dans de la parole, et comprenant en outre :
   des moyens de transducteur, couplés aux moyens de mélange, pour convertir la séquence numérique en un son dans une synthèse de parole.

10. Appareil pour assembler par concaténation une première trame numérique de N échantillons ayant des amplitudes respectives représentant un premier segment de son et une deuxième trame numérique de M échantillons ayant des amplitudes respectives représentant un deuxième segment de son, comprenant :

   une mémoire tampon pour stocker les échantillons des première et deuxième trames numériques ;
   des moyens, couplés à la mémoire tampon, pour déterminer un point de juxtaposition pour les première et deuxième trames numériques en réponse à des amplitudes d'échantillons dans les première et deuxième

trames numériques ;
des moyens de mélange, couplés à la mémoire tampon et aux moyens de détermination, pour calculer une séquence numérique représentant une concaténation des premier et deuxièmes segments de son en réponse à la première trame, à la deuxième trame et au point de juxtaposition ; et
des moyens de transducteur, couplés aux moyens de mélange, pour convertir la séquence numérique en un son.

11. Appareil selon la revendication 10, dans lequel les moyens de détermination comprennent :

    des premiers moyens pour calculer une trame étendue en réponse à la première trame numérique ;
    des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique, et définir le point de juxtaposition comme étant un échantillon dans le sous-ensemble.

12. Appareil selon la revendication 11, dans lequel la trame étendue comprend une concaténation de la première trame numérique avec une réplique de la première trame numérique.

13. Appareil selon la revendication 11 ou 12, dans lequel le sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique est un sous-ensemble ayant une différence moyenne minimale d'amplitude entre les échantillons dans le sous-ensemble, et le point de juxtaposition est un premier échantillon dans le sous-ensemble.

14. Appareil selon l'une quelconque des revendications 10 à 13, dans lequel les moyens de détermination comprennent :

    des premiers moyens pour calculer une trame étendue comprenant une concaténation à lissage de discontinuité de la première trame numérique avec une réplique de la première trame numérique ;
    des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue ayant une différence moyenne minimale d'amplitude entre les échantillons dans le sous-ensemble et la deuxième trame numérique, et définir le point de juxtaposition comme étant un premier échantillon dans le sous-ensemble.

15. Appareil selon l'une quelconque des revendications 10 à 14, dans lequel les moyens de mélange comprennent :

    des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et
    des moyens pour combiner la deuxième trame numérique avec un deuxième ensemble d'échantillons extrait de la première trame numérique et du point de juxtaposition, avec une mise en valeur du deuxième ensemble dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin, afin de produire un deuxième segment de la séquence numérique.

16. Appareil selon la revendication 14, dans lequel les moyens de mélange comprennent :

    des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et
    des moyens pour combiner la deuxième trame numérique avec le sous-ensemble de la trame étendue, avec une mise en valeur du sous-ensemble de la trame étendue dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin, afin de produire un deuxième segment de la séquence numérique.

17. Appareil selon la revendication 16, dans lequel les première et deuxième trames numériques représentent respectivement des fins et des commencements de diphones adjacents dans de la parole, et les moyens de transducteur produisent une parole synthétisée.

18. Appareil pour synthétiser de la parole en réponse à un texte, comprenant :

    des moyens (21) pour traduire un texte en une séquence de codes de segment de son ;
    des moyens (23) répondant aux codes de segment de son dans la séquence, pour décoder la séquence de codes de segment de son afin de produire des chaînes de trames numériques d'une pluralité d'échantillons

représentant des sons pour des codes correspondants de segment de son dans la séquence, dans lequel les chaînes identifiées de trames numériques ont des commencements et des fins ;

des moyens (24) pour réunir par concaténation une première trame numérique à la fin d'une chaîne identifiée de trames numériques d'un code particulier de segment de son dans la séquence, avec une deuxième trame numérique au début d'une chaîne identifiée de trames numériques d'un code adjacent de segment de son dans la séquence, afin de produire une séquence de données de parole, comprenant

une mémoire tampon pour stocker les échantillons de première et deuxième trames numériques ;

des moyens, couplés à la mémoire tampon, pour déterminer un point de juxtaposition pour les première et deuxième trames numériques en réponse à des amplitudes d'échantillons dans les première et deuxième trames numériques ; et

des moyens de mélange, couplés à la mémoire tampon et aux moyens de détermination, pour calculer une séquence numérique représentant une concaténation des premier et deuxièmes segments de son en réponse à la première trame, à la deuxième trame et au point de juxtaposition ; et

un transducteur audio (27), couplé aux moyens de concaténation, pour produire une parole synthétisée en réponse à la séquence de données de parole.

19. Appareil selon la revendication 18, comprenant en outre :

des moyens (25, 26) répondant aux codes de segment de son pour ajuster la hauteur et la durée des chaînes identifiées de trames numériques dans la séquence de données de parole.

20. Appareil selon la revendication 18 ou 19, dans lequel les moyens de détermination comprennent :

des premiers moyens pour calculer une trame étendue en réponse à la première trame numérique ;

des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique, et définir le point de juxtaposition comme étant un échantillon dans le sous-ensemble.

21. Appareil selon la revendication 20, dans lequel la trame étendue comprend une concaténation de la première trame numérique avec une réplique de la première trame numérique.

22. Appareil selon la revendication 20 ou 21, dans lequel le sous-ensemble de la trame étendue qui coïncide relativement bien avec la deuxième trame numérique comprend un sous-ensemble ayant une différence moyenne minimale d'amplitude avec les échantillons dans le sous-ensemble, et le point de juxtaposition comprend un premier échantillon dans le sous-ensemble.

23. Appareil selon l'une quelconque des revendications 18 à 22, dans lequel les moyens de détermination comprennent :

des premiers moyens pour calculer une trame étendue comprenant une concaténation à lissage de discontinuité de la première trame numérique avec une réplique de la première trame numérique ;

des deuxièmes moyens pour trouver un sous-ensemble de la trame étendue ayant une différence moyenne minimale d'amplitude entre les échantillons dans le sous-ensemble et la deuxième trame numérique, et définir le point de juxtaposition comme étant un premier échantillon dans le sous-ensemble.

24. Appareil selon l'une quelconque des revendications 18 à 23, dans lequel les moyens de mélange comprennent :

des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et

des moyens pour combiner la deuxième trame numérique avec un deuxième ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition, avec une mise en valeur du deuxième ensemble dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin, afin de produire un deuxième segment de la séquence numérique.

25. Appareil selon la revendication 23, dans lequel les moyens de mélange comprennent :

des moyens pour fournir un premier ensemble d'échantillons extraits de la première trame numérique et du point de juxtaposition sous la forme d'un premier segment de la séquence numérique ; et

des moyens pour combiner la deuxième trame numérique avec le sous-ensemble de la trame étendue, avec

une mise en valeur du sous-ensemble de la trame étendue dans un échantillon de début et une mise en valeur de la deuxième trame numérique dans un échantillon de fin afin de produire un deuxième segment de la séquence numérique.

5     26. Appareil selon l'une quelconque des revendications 18 à 25, dans lequel les codes de segment de son représentent des diphones de parole, et les première et deuxième trames numériques représentent respectivement des fins et des débuts de diphones adjacents dans de la parole.

10

15

20

25

30

35

40

45

50

55

FIG.−1

```
                    ┌─────────────────────────┐
                    │   RECEIVE INPUT TEXT     │──────⌐20
                    └─────────────────────────┘
                            │              └──────────────────┐
        ┌─────────────────────────┐          ┌─────────────────────────┐
        │      TRANSLATE TO        │          │       GENERATE          │⌐22
  21──  │     DIPHONE STRINGS      │          │      INTONATION         │
        └─────────────────────────┘          │     CONTROL DATA        │
                    │                         └─────────────────────────┘
        ┌─────────────────────────┐                      │
        │       DECOMPRESS         │                      │
  23──  │   DIPHONE STRINGS TO     │                      │
        │ GENERATE VQ DATA FRAMES  │                      │
        └─────────────────────────┘                      │
                    │                                     │
        ┌─────────────────────────┐                      │
  24──  │     BLEND  DIPHONE       │                      │
        │     VQ  DATA FRAMES      │                      │
        └─────────────────────────┘                      │
                    │                                     │
        ┌─────────────────────────┐                      │
        │        ADJUST            │                      │
  25──  │   DURATION OF DIPHONE    │◄─────────────────────┤
        │     VQ DATA FRAMES       │                      │
        └─────────────────────────┘                      │
                    │                                     │
        ┌─────────────────────────┐                      │
        │        ADJUST            │                      │
  26──  │    PITCH OF DIPHONE      │◄─────────────────────┘
        │     VQ DATA FRAMES       │
        └─────────────────────────┘
                    │
        ┌─────────────────────────┐
  27──  │   SUPPLY SPEECH DATA     │
        │    TO AUDIO OUTPUT       │
        └─────────────────────────┘
```

TEXT – TO – SPEECH   CODE

# FIG.–2

Diphone Record

| Left Diphone | Right Diphone |
|---|---|
| Left Pitch Period Count | Right Pitch Period Count |
| Pointer to Left Pitch Period | Pointer to Right Pitch Period |
| Pointer to Left Demi Data | Pointer to Right Demi Data |

30 → Left Diphone
31 → Right Diphone
32 → Left Pitch Period Count
33 → Pointer to Left Pitch Period
34
35
36

| $LP_0$ |
|---|
| $LP_1$ |
| |
| $LP_{NL-1}$ |

Pitch Table

| $LFRAME_0$ |
|---|
| $LFRAME_1$ |
| |
| $LFRAME_{ML-1}$ |

VQ Compressed Speech Records

| $RFRAME_0$ |
|---|
| $RFRAME_1$ |
| |
| $RFRAME_{MR-1}$ |

VQ Compressed Speech Records

| $RP_0$ |
|---|
| $RP_1$ |
| |
| $RP_{NR-1}$ |

Pitch Table

FIG.−3

FIG.—4

FIG.−5



FIG.−6

FRAME
DECODER — 200

Decode
Parameters
$G$, $\beta$, $P_{opt}$,
VQ string — 201

ACCESS AND
CONCATENATE
QUANTIZATION
VECTORS FOR
VQ STRING — 203

125 —

$QV_0$
$QV_1$
$QV_2$
•
•
•
$QV_{255}$

Decode
Residual
Signal  $r'_n$ — 202

Inverse Pitch
Filter
$y'_n$ — 204

Sysnthesis Pitch
Buffer Update
SPBUF — 205

Inverse Linear
Predictive
Filtering
$x'_n$ — 206

OUTPUT
SPEECH

FIG.-7

```
┌─────────────────────────┐
│   RECEIVE LEFT AND      │───  300
│    RIGHT DIPHONE        │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│   STORE LAST FRAME      │
│   OF LEFT DIPHONE       │───  301
│   IN BUFFER $L_n$       │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│   STORE FIRST FRAME     │
│   OF RIGHT DIPHONE      │───  302
│   IN BUFFER $R_n$       │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│   REPLICATE AND         │───  303
│   CONCATENATE $L_n$     │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│      SMOOTH             │
│   DISCONTINUITY         │───  304
│      ( $El_n$ )         │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│  FIND OPTIMUM MATCH OF  │
│   $R_n$ TO $El_n$       │───  305
│      ( $P_{opt}$ )      │
└─────────────────────────┘
             │
             ▼
┌─────────────────────────┐
│       BLEND             │
│   $El_n$ AND $R_n$      │───  306
│   WITH $P_{opt}$        │
└─────────────────────────┘
```
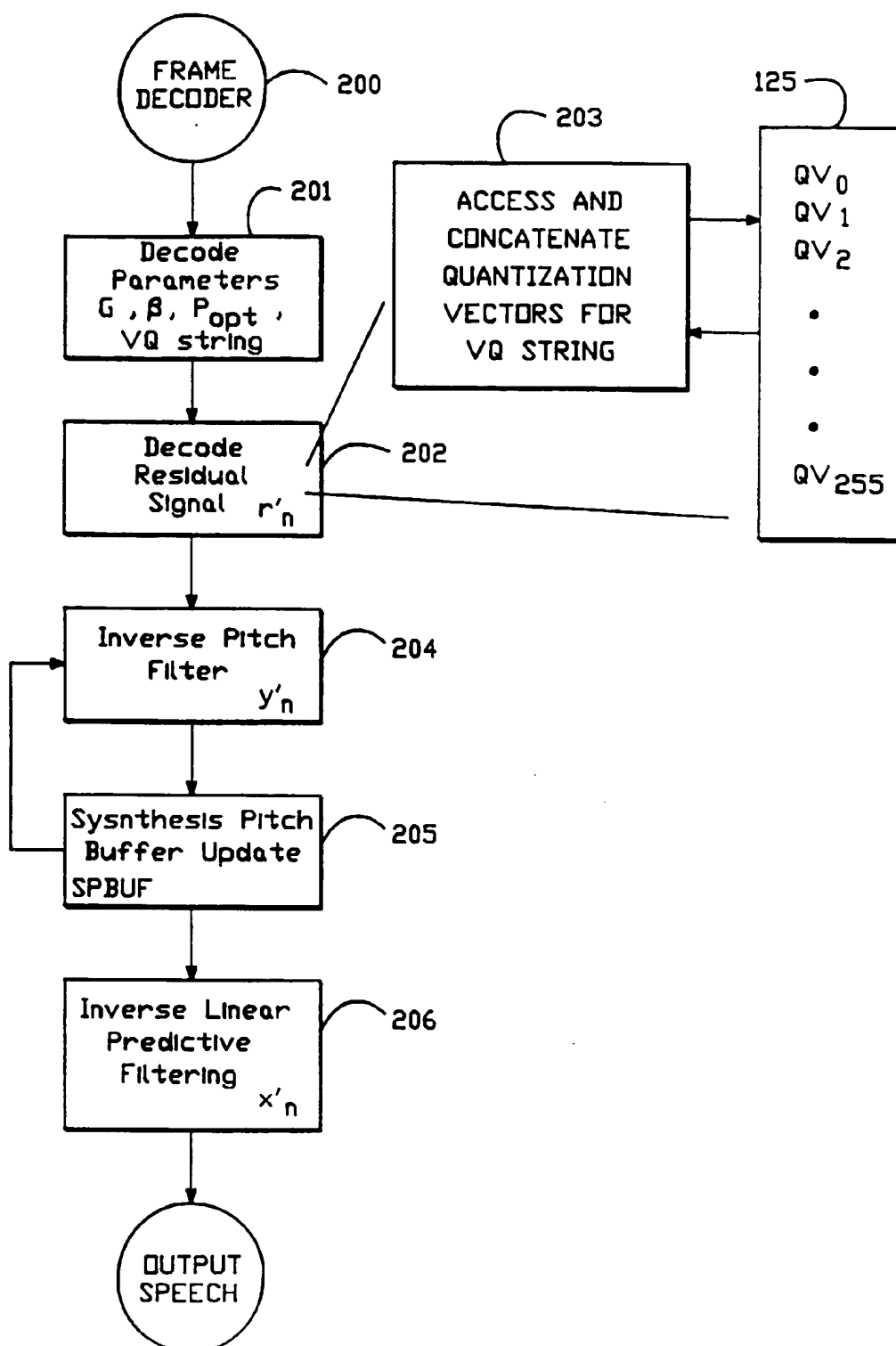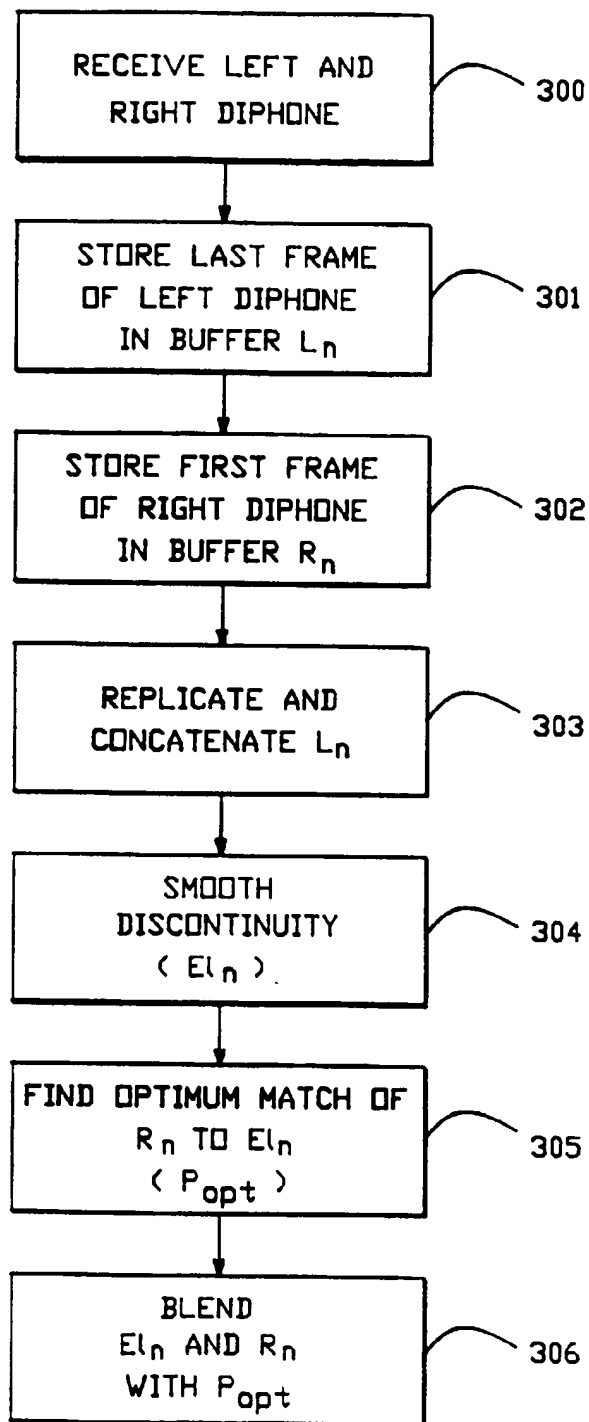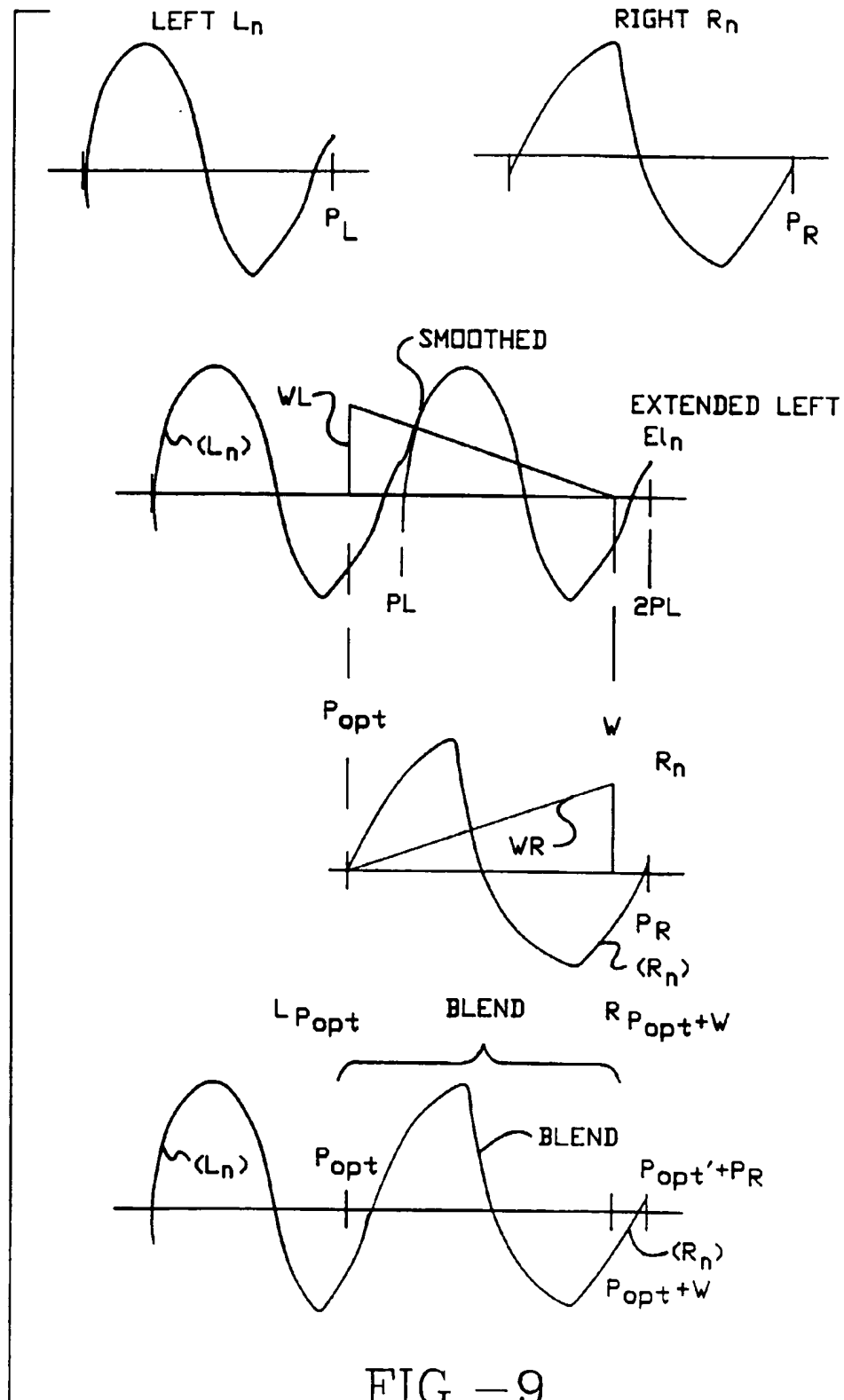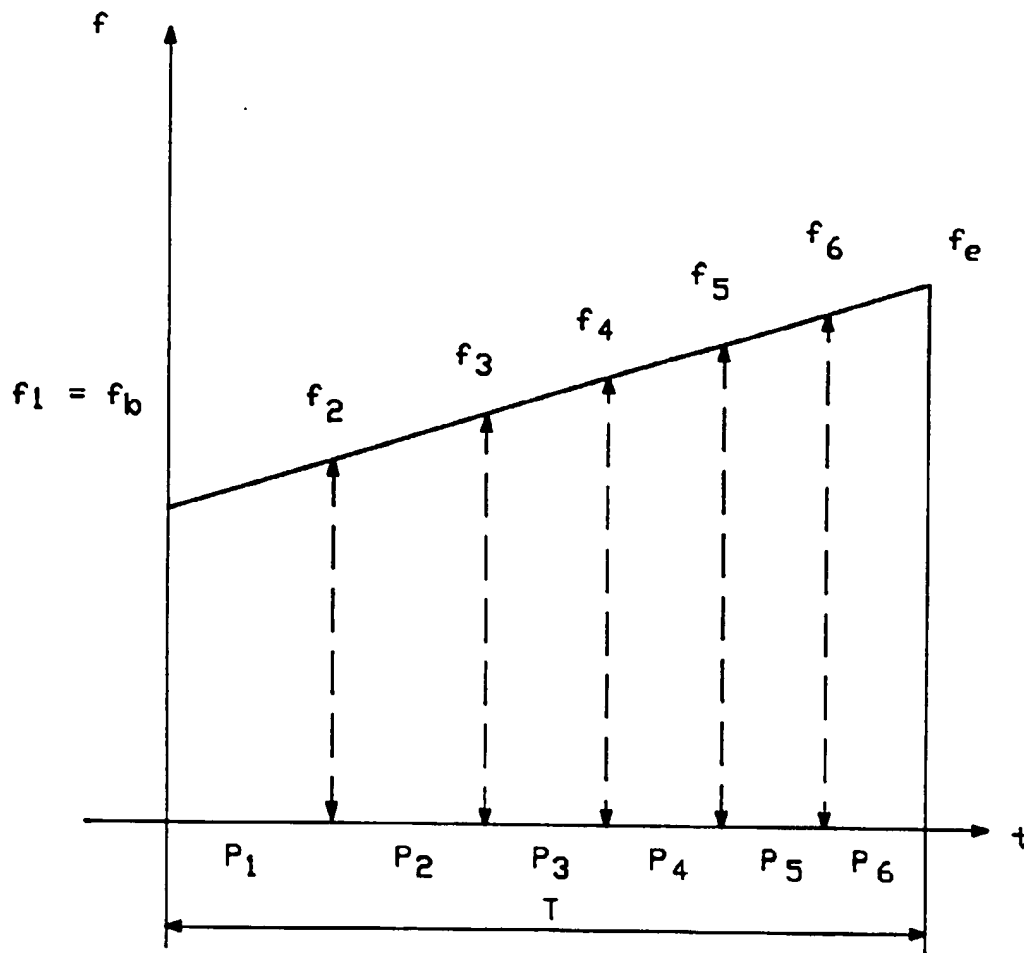
# FIG.−8

FIG.−9

NOTES:

T = Desired duration of a phoneme

$f_b$ = Desired Begining Pitch in Hz

$f_e$ = Desired Ending Pitch in Hz

P1, P2, ..., P6 are the desired pitch period in No. of Samples corresponding to the frequencies f1,f2,...f6.

Relationship between Pi and fi:

Pi = Fs/fi, where Fs is the Sampling frequency.

FIG.-10

INCREASE PITCH PERIOD
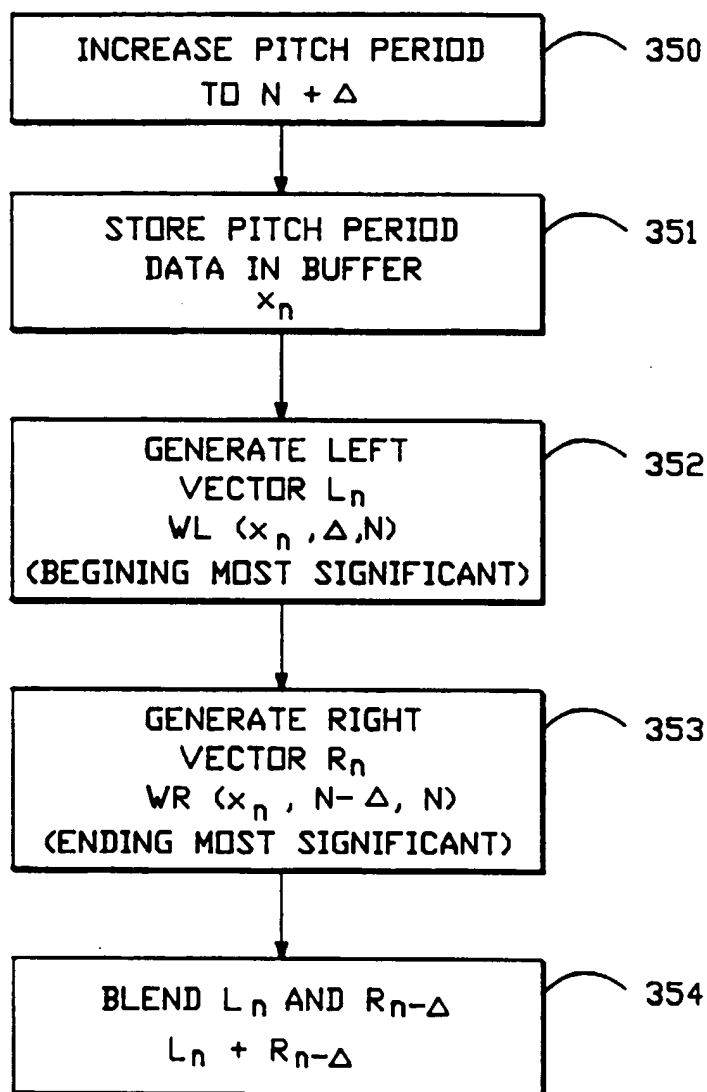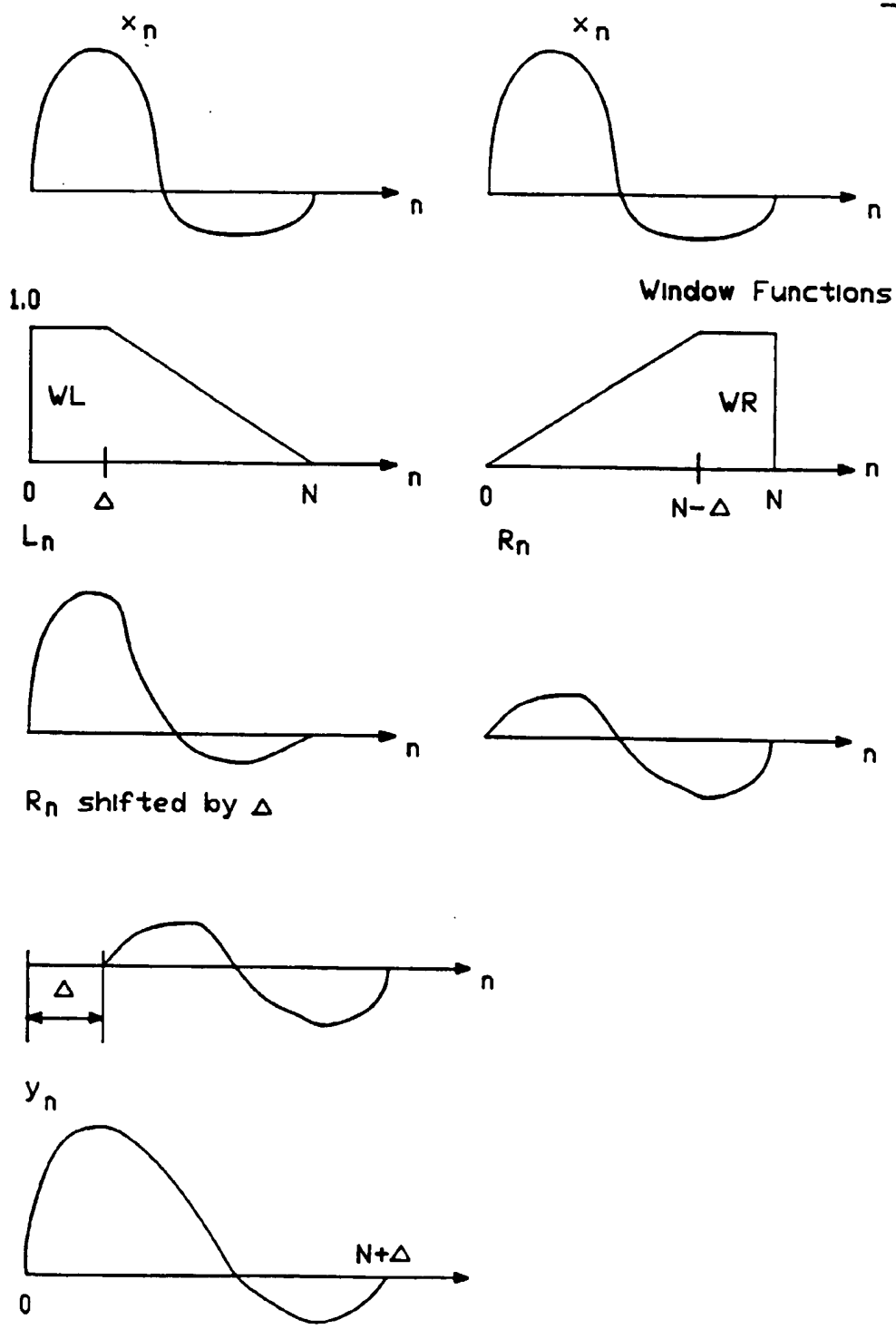TO $N + \Delta$ — 350

STORE PITCH PERIOD
DATA IN BUFFER
$x_n$ — 351

GENERATE LEFT
VECTOR $L_n$
$WL (x_n, \Delta, N)$
(BEGINING MOST SIGNIFICANT) — 352

GENERATE RIGHT
VECTOR $R_n$
$WR (x_n, N-\Delta, N)$
(ENDING MOST SIGNIFICANT) — 353

BLEND $L_n$ AND $R_{n-\Delta}$
$L_n + R_{n-\Delta}$ — 354

# FIG.-11

FIG.−12

DECREASE PITCH PERIOD
TO N – $\Delta$      400

STORE TWO PITCH
PERIODS IN BUFFER
$x_n$      401

GENERATE LEFT VECTOR
$L_n = WL\ (x_n,\ N_l,\ W)$
(BEGINING MOST SIGNIFICANT)      402

GENERATE RIGHT VECTOR
$L_n = WR\ (x_n,\ N_l + N_r,\ W)$
(ENDING MOST SIGNIFICANT)      403

BLEND $L_n$ AND $R_{n+\Delta}$
$L_n + R_{n+\Delta}$      404

# FIG.–13

FIG.—14

INSERT PITCH PERIOD
BETWEEN $L_n$ AND $R_n$ — 450

STORE $L_n$ AND $R_n$
IN BUFFER — 451

GENERATE LEFT
VECTOR WL $(L_n)$
(ENDING MOST SIGNIFICANT) — 452

GENERATE RIGHT
VECTOR WR $(R_n)$
(BEGINING MOST SIGNIFICANT) — 453

BLEND WR $(L_n)$ AND WR $(R_n)$
TO INSERTED PERIOD $x_n$ — 454

CONCATENATE
$L_n \longrightarrow x_n \longrightarrow R_n$ — 455

# FIG.–15

FIG.−16

INSERT PITCH PERIOD $R_n$ WHICH FOLLOWS $L_n$ — 500

STORE $L_n$ AND $R_n$ IN BUFFER — 501

GENERATE LEFT VECTOR WL ($L_n$) (BEGINING MOST SIGNIFICANT) — 502

GENERATE RIGHT VECTOR WR ($R_n$) (ENDING MOST SIGNIFICANT) — 503

BLEND WL ($L_n$) AND WR ($R_n$) TO CREATE RESULTING $L'_n$ — 504

REPLACE $L_n \longrightarrow R_n$ WITH $L'_n$ IN PITCH PERIOD STRING — 505

FIG. –17

$L_n$ $R_n$

N N

Weighting Function

1.0

WL WR

N N

$WL(L_n)$ 0 $WR(R_n)$ N

0 N

+

$W(L_n) + W(R_n)$

$L'_n$

FIG.−18